COMMODORE 64

INTRODUZIONE AL BASIC. PARTE 1

COD. 6024

LA SERIE COMPLETA DI PROGRAMMAZIONE AUTODIDATTICA

by Andrew Colin

Questo corso è la Parte 1 di una serie intesa ad aiutare ad imparare qualsiasi aspetto della programmazione del computer Commodore **64.** Il presente corso tratta i principi di programmazione e tutte le funzioni elementari del linguaggio di programmazione BASIC.
Esso si compone di 2 parti:

1. Un testo autodidattico diviso in 15 lezioni o "unità", ciascuna delle quali tratta un importante aspetto della programmazione.

2. 2 cassette di nastro con una raccolta di programmi **64** che aiuteranno nello studio delle unità.

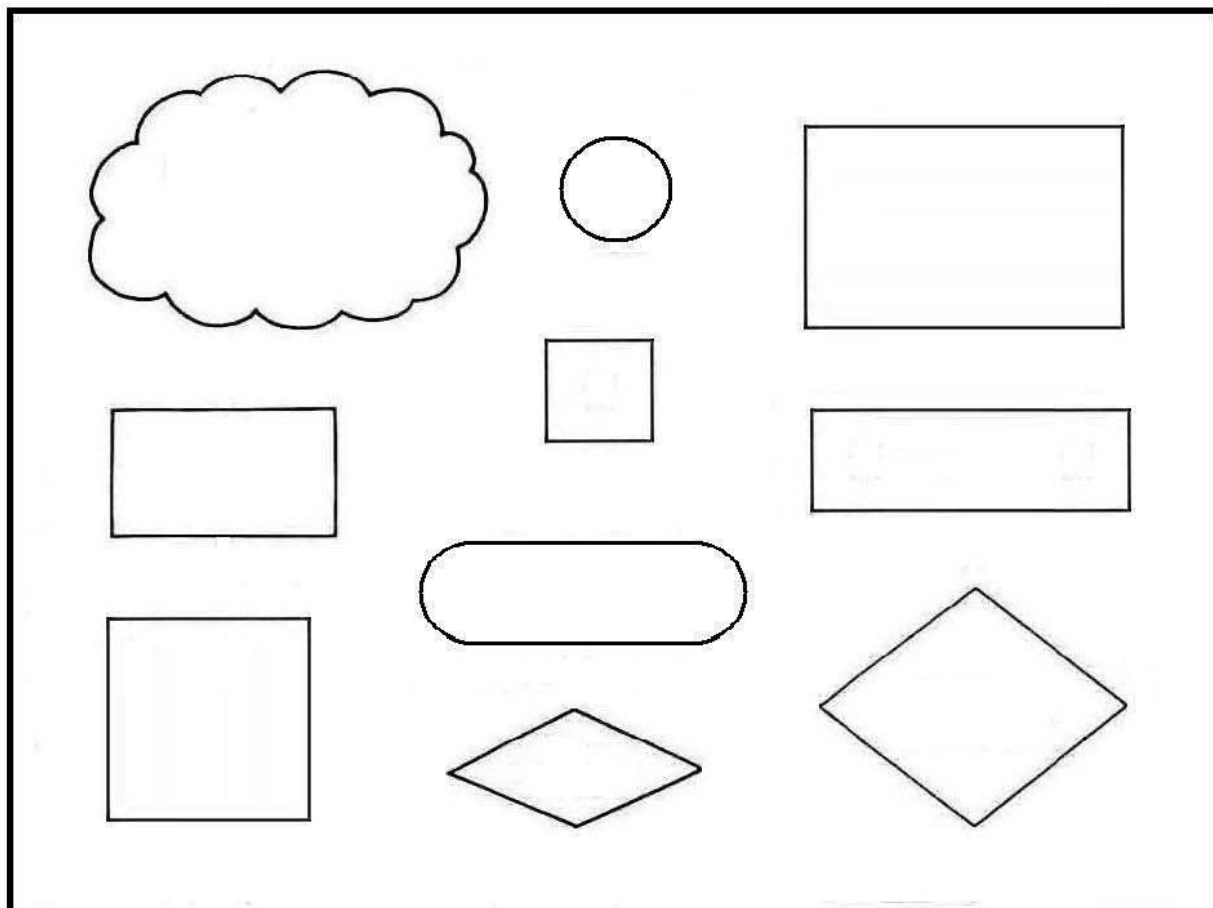
Inoltre viene fornita una squadretta per aiutare il disegno degli schemi di flusso.

Va notato che questo corso insegna a scrivere utili e divertenti programmi per il **64** ma *non* illustra tutti gli aspetti del linguaggio BASIC. Le caratteristiche più avanzate del BASIC sono spiegate a fondo e discusse nel secondo volume di questa serie.

INDICE

La dima è una squadretta di plastica che serve da guida per disegnare. Nella confezione originale la dima per disegnare gli schemi di flusso era di plastica e di dimensioni circa 8x14 cm.
Qui ne viene presentata una ricavata dagli schemi di flusso presentati nel libro.

Si può stampare la dima qui presente e incollarla su un cartoncino o su un rettangolino di legno compensato o di balsa dura. Quando è asciutto, si può ritagliare il cartoncino e la balsa con un taglierino oppure segare il compensato con il seghetto e ricavare la propria dima.

Questa pagina non è presente nell'edizione originale.

INTRODUZIONE

Congratulazioni e benvenuti a questo corso di programmazione. Il **64** è una superba macchina in grado di eseguire giochi e produrre immagini e suoni eccitanti e brillanti sul televisore ma è anche, a tutti gli effetti, un moderno, completo computer.
I computer sono eccezionalmente versatili, molto di più di qualsiasi cosa salvo gli esseri umani. Il **64**, per esempio, può essere trasformato in una macchina per giocare o per insegnare, in un calcolatore, in un aiuto per l'handicappato, in una macchina per eseguire registrazioni finanziarie e controllo delle scorte, in un monitor per un paziente in un'unità di cura intensiva, in un'unità di controllo per un processo industriale o in un computer scientifico usato dai tecnici per progettare fabbricati, centrali elettriche e aerei.
I computer e i sistemi che essi controllano stanno entrando incessantemente nella nostra vita di ogni giorno e sono già presenti in numerosi dispositivi tipo semafori, registratori di cassa, terminali bancari. Questa tendenza continuerà per la maggior parte della nostra vita. Il mondo sta attraversando una rivoluzione del computer che sarà altrettanto profonda, per quanto riguarda gli effetti, di quella che è stata ai suoi tempi la rivoluzione industriale.
La rivoluzione del computer non può essere fermata ma tutti possiamo, volendo, avere qualche influenza sul modo in cui essa si svolge. Gli esseri umani si stanno dividendo in due categorie, i passeggeri e i piloti. I passeggeri non si preoccupano di nulla e lasciano che le cose vadano per il loro verso: al limite possono divertirsi ad usare i prodotti computerizzati od odiare i computer o fare entrambe le cose. Spesso rendono noti i rispettivi punti di vista ma senza alcun effetto reale — essi non possono cioè raggiungere la cosiddetta "stanza dei bottoni" e non saprebbero usarli, se lo potessero.
I piloti, per contro, hanno il controllo dell'intera rivoluzione. Essi inventano nuovi tipi di computer e pensano ai modi originali e utili per usarli. I piloti hanno una grave responsabilità dato che tocca a loro guidare il mondo verso la pace, la libertà e l'abbondanza, allontanandolo dalla società da incubo spesso illustrata nella fantascienza.
Che cosa distingue un pilota da un passeggero? Una sola cosa: la conoscenza del modo in cui funziona il computer. Naturalmente ci sono diversi livelli di conoscenza. La maggior parte delle persone sanno come usare una macchina per giocare agli "invasori spaziali" quantunque essi non sappiano spiegarne il meccanismo. (Sì, — c'è di mezzo un computer anche in "Invasori spaziali"). Il livello di cui stiamo parlando è molto più profondo. È così completo e approfondito che è possibile far sì che un computer faccia qualsiasi cosa — giocare, insegnare o rendersi utile nelle applicazioni industriali o medicali.
Per avere questo potere sul proprio computer, per trasformarlo in uno schiavo veloce, accurato, obbediente e volonteroso, occorre essere in grado di *programmarlo*. La programmazione è il segreto per diventare un pilota.
Questo corso si occupa di programmazione. Esso si riferisce specificamente al Commodore **64** ma una volta imparata la programmazione col **64**, sarà semplice trasferirne i concetti base a qualsiasi altro computer, grande o piccolo che sia.
Più si procede nella programmazione e più questa risulterà facile. Tutti possono imparare a programmare con un po' di buona volontà. Non occorre conoscere a fondo la matematica ma bensì sarà utile trovarsi un posto tranquillo per leggere, pensare e usare il **64**. Sarà inoltre bene riservarsi tempo a sufficienza per completare il corso. È inutile correre!
Il corso è diviso in quindici "unità" o capitoli. Ciascuna unità comprenderà in media uno o due "compiti". La maggior parte delle unità comprendono un po' di lettura, un certo lavoro pratico sul **64**, un po' di programmazione e un questionario per rendersi conto se si è compreso il capitolo. Ogni unità contiene qualche "esperimento" che è possibile svolgere.
Quando l'unità pone domande, è previsto generalmente spazio per scrivere le risposte. È bene sfruttare questa possibilità. Scrivere con una matita morbida e tenere a disposizione una gomma in modo da poter cancellare le risposte.
La programmazione è un argomento i cui concetti dipendono strettamente l'uno dall'altro. Quelli imparati nelle prime unità sono citati e usati nelle ultime, senza ulteriore spiegazione. Per esempio, non si potrà rischiare di affrontare l'unità 10 a meno che non si siano lette tutte le unità da 1 a 9. Ciò rende importante seguire le

unità nell'ordine indicato.
Quando si inizia a lavorare su una nuova unità, cominciare leggendo rapidamente dall'inizio fino alla fine. Non si acquisiranno così molti dettagli ma ci si formerà un'idea del tipo di argomento che verrà trattato.
Successivamente, sarà bene riesaminare l'unità in dettaglio. Ogni parte è importante e quelle che sembrano più ostiche lo sono di più. Non saltare nulla, ma cercare invece di capire ogni punto. Quando si pensa di aver imparato qualche cosa, provare a ripeterselo con proprie parole. Non arrabbiarsi se si scopre che è necessario rileggere le varie parti dell'unità parecchie volte o ritornare indietro ad un'unità precedente per chiarire qualche punto oscuro. Si tratta di un fatto abbastanza normale con un argomento tecnico.
Programmare è come suonare uno strumento musicale: si può imparare solo facendo pratica. È quindi opportuno svolgere tutti i problemi di programmazione nel corso. Non appena è possibile, iniziare a creare e a risolvere problemi per conto proprio.
Una volta completato il corso si sarà in grado di usare il **64** per molti scopi diversi — ad esempio fagli svolgere delle prove o dei quiz, fargli svolgere giochi di propria invenzione ed addirittura trovarlo utile per fare somme e tenere i conti. I giochi o altre applicazioni possono comprendere immagini colorate e disegnate dall'utente nonchè suoni per sottolinearne il significato — meravigliosi suoni o rudi rumori!
La programmazione è in ogni caso un argomento molto vasto che non si può esaurire in un solo corso. Dopo un certo periodo di tempo, si vorrà probabilmente approfondirne i concetti. Si vorranno risolvere problemi più complicati oppure usare il **64** come unità di controllo per un plastico ferroviario o per un centralino telefonico privato.

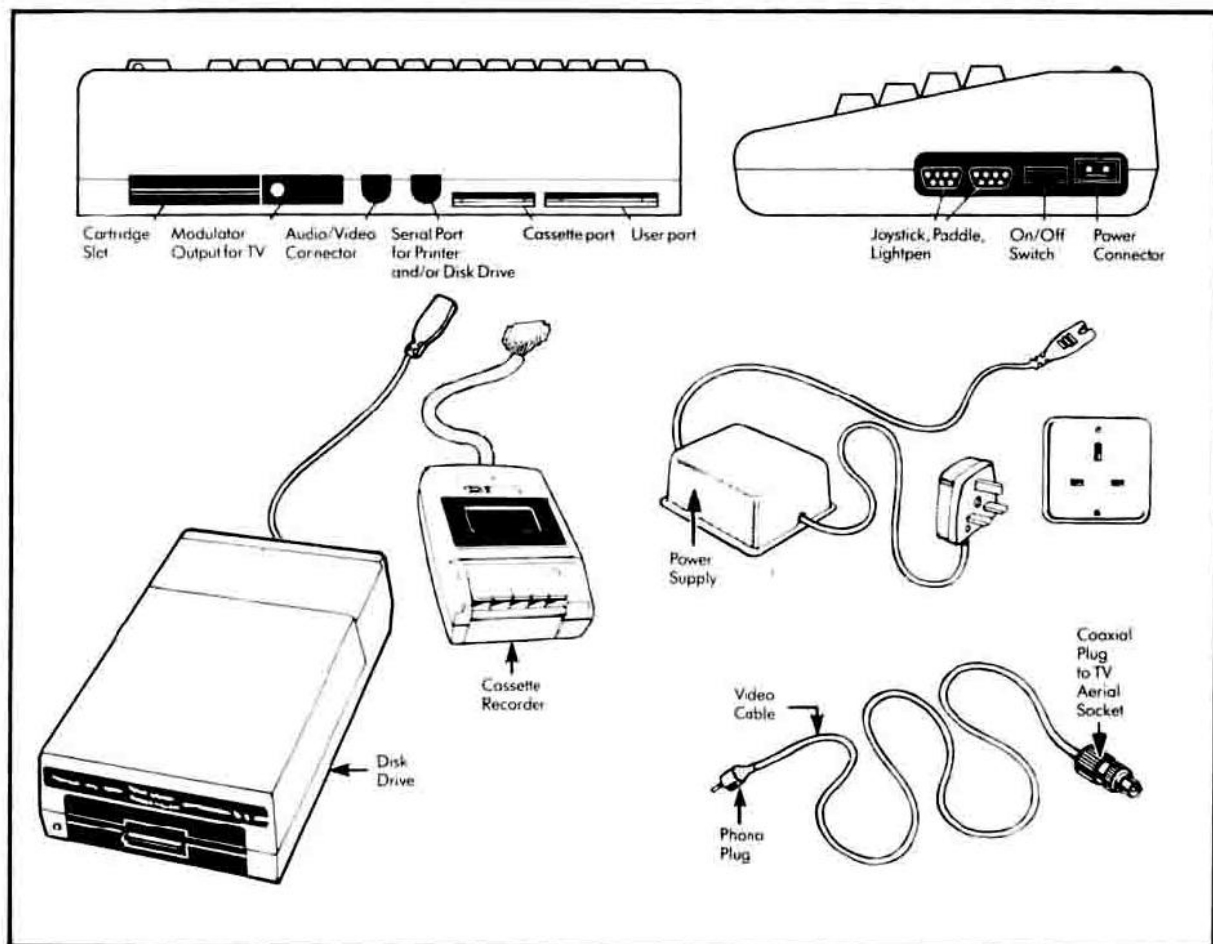
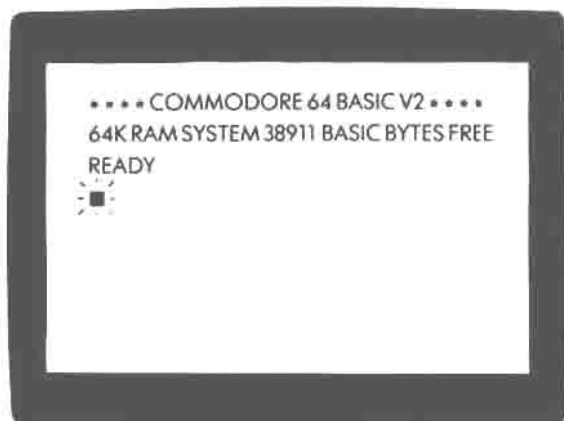
Il secondo volume di questa serie intitolato: Introduzione al BASIC Parte 2 sarà un ulteriore aiuto per approfondire queste conoscenze.
Ma ora basta con le parole. È ora di iniziare l'Unità 1. Buona fortuna!

UNITA':1

Questa unità aiuta a prendere contatto con il **64,** spiegando un certo numero di argomenti piuttosto ordinari, questioni pratiche che spesso comportano seri problemi quando la gente compra il suo primo computer.
Per imparare la programmazione occorre avere l'ambiente adatto. Trovarsi quindi un posto comodo e riservarsi adeguati periodi (almeno due ore) in un periodo del giorno in cui non si è troppo stanchi per potersi concentrare. Fare tutto ciò che è possibile per non essere disturbati — mettere un cartello sulla porta, staccare il ricevitore del telefono e dire a tutti che si è occupati: non c'è nulla che renda la programmazione più difficile che le interruzioni costanti e la mancanza di concentrazione.
Se si è installato il **64** e lo si è già usato, è possibile saltare direttamente all'esperimento 1.1, altrimenti leggere rapidamente l'unità anche se si conosce già l'argomento; un ripasso può sempre essere utile.
Innanzitutto predisporre l'apparecchiatura e collegarla alla rete. Il **64,** l'alimentatore e la cassetta od il floppy vanno disposti sulla tavola o sulla scrivania e il televisore deve essere disposto ad almeno 2 metri di distanza se si tratta di un piccolo apparecchio o anche a distanza maggiore se si tratta di uno a grande schermo. Le immagini e il testo prodotti dal **64** sono abbastanza ampi per poter essere letti alla normale distanza di osservazione. Si scoprirà che lavorare con uno schermo troppo ravvicinato è irritante e stancante.

1

Le varie unità si collegano come indicato nel diagramma.
Tutte le spine devono entrare nelle rispettive prese con una pressione leggera ma costante. Non forzare mai ma osservare accuratamente la disposizione dei piedini e delle spine e delle prese prima di unirli.
Il **64** è una macchina estremamente robusta ma spine e prese si consumano o si rompono se sono utilizzate male o un numero eccessivo di volte. Una volta che il **64** è predisposto, conviene lasciarlo indisturbato per quanto possibile.
Sia il **64** che il televisore, possono essere alimentati da un unico cavo di prolunga con prese doppie. Tale cavo consente massima libertà per decidere come disporre il sistema.
Ora è possibile procedere.
Accendere il televisore e scegliere un canale che non è normalmente usato per la ricezione delle trasmissioni. (Ad esempio, se l'apparecchio è sintonizzato per ricevere i canali 1, 2 e 3, è possibile usare il canale 4 per il **64**). L'apparecchio farà un certo rumore e può essere necessario ridurre il volume del suono.
Successivamente accendere il **64** usando l'interruttore sul lato destro. Se tutto funziona regolarmente, dovrebbe accendersi la spia rossa POWER e, a meno che non si sia molto fortunati, il televisore non presenterà ancora alcuna immagine.
Tornare quindi al televisore e regolare la sintonia del canale scelto. Il metodo esatto di sinto-

nia varia secondo la marca dell'apparecchio e in ogni caso è spiegato nel libretto di istruzioni; nella maggior parte dei casi c'è una piccola manopola o una vite corrispondenti a ciascun canale. Talvolta le regolazioni di sintonia sono nascoste dietro un piccolo pannello.
Una volta messa a punto la sintonia, dovrebbe comparire improvvisamente un'immagine:

Il quadrato centrale è blu con un bordo azzurro. Può darsi che sia necessario regolare la sintonizzazione verticale e la linearità per ottenere un'immagine stabile.

Se non si ottiene questa immagine o se l'immagine appare soltanto in bianco e nero, spegnere il **64** per alcuni secondi, quindi riaccenderlo.
Se s'incontrano difficoltà, controllare i seguenti punti:

- Il televisore funziona? Provare con la ricezione su un programma normale e farlo riparare se è il caso.
- La spia di accensione del **64** è illuminata? In caso contrario, controllare:
  (a) Che non ci sia una mancanza generale di corrente.
  (b) Che sia collegato qualche altro dispositivo (lampada da tavolo o asciugacapelli) alla presa in uso. In caso contrario provare a cambiare il fusibile nella spina del cavo di prolunga.
  (c) Che la spia nell'alimentatore del **64** sia intatta (provare con un nuovo fusibile).
  (d) Che l'alimentatore sia saldamente collegato al **64.**
- Il **64** è correttamente collegato alla presa d'antenna sul televisore?

Se il sistema non funziona ancora, riportarlo al rivenditore per farsi consigliare ed eventualmente farlo riparare.
Il messaggio che compare sullo schermo si compone di un certo numero di caratteri, comprendenti lettere, numeri e simboli tipo ★. Questi caratteri sono sempre della stessa misura e quando lo schermo è completamente pieno ne contiene 1000.
La prima riga sullo schermo identifica il prodotto: un sistema BASIC progettato e costruito dalla Commodore Business Machines. V2 è un numero di versione che può variare di volta in volta.
Il messaggio sulla riga successiva dello schermo dice quanta memoria c'è nella macchina. Ogni computer ha bisogno di una "memoria" per accogliere i dettagli del lavoro che deve eseguire. La memoria si misura in "byte", ciascuno dei quali può contenere un simbolo o *carattere* d'informazione. Maggiore è la quantità di memoria e più complesse sono le funzioni che la macchina può svolgere.
Se il numero dello schermo è minore di 38911, significa che il **64** si è guastato. Deve essere quindi riportato al rivenditore per la riparazione.
La terza riga dice che il **64** è ora pronto a obbedire ai comandi che si batteranno sulla tastiera.
La riga successiva visualizza un quadratino lampeggiante detto *cursore*. Quando si batte un comando sulla tastiera, il cursore mostra in anticipo esattamente dove ciascun carattere verrà visualizzato. Per esempio, provare quanto segue:

PRINT 5 + 8 RETURN

(Ciò richiede 10 operazioni sui tasti:

P R I N T SPACE 5 + 8 e la pressione del tasto RETURN. Questo è il grande tasto che si trova alla destra della tastiera). (Prima di iniziare la battitura, premere il tasto SHIFT LOCK per assicurarsi che non sia bloccato). Man mano che si batte ciascun simbolo (salvo RETURN) questo compare sullo schermo e il cursore si sposta di una posizione.

La prima funzione del tasto RETURN è di far sì che il computer esegua un'istruzione. In questo caso si tratta di stampare (cioè visualizzare) il risultato della somma di 5 e 8!

Per poter fare qualcosa di utile, il COMMODORE 64 deve avere un programma.
I programmi sono memorizzati su cassette o su dischetti e il primo esperimento consentirà di far pratica nel caricare nel **64** un programma da nastro o da disco. Se si dispone di un registratore a cassetta, procedere con le istruzioni seguenti. Se per contro si dispone di un'unità a dischetti, saltare all'esperimento 1.2 studiato specificamente per questa eventualità.

Caricamento di programmi da nastro.
Seguire attentamente le istruzioni:

1. Assicurarsi che l'unità a cassetta sia collegata al **64.**
2. Premere STOP sull'unità a cassetta.
3. Aprire lo scomparto dell'unità a cassetta, estrarre il nastro che potrebbe esservi inserito e inserire il nastro TESTCARD — con l'etichetta nella parte superiore e con la finestra del nastro rivolta verso l'operatore. Chiudere lo scomparto. Se non si chiude, non forzare ma assicurarsi di aver inserito il nastro nel modo corretto.
4. Premere il tasto REWIND sul registratore. Osservare la cassetta attraverso la finestra e se si vedono le bobine girare, attendere fino a che non si fermano. Assicurarsi di portarsi all'inizio del nastro.
5. Premere il tasto stop sul registratore.
6. Battere ora il seguente messaggio:

   LOAD "TESTCARD" RETURN

   Ciò richiede in tutto 15 operazioni sui tasti, contando le " come un unico carattere. Per produrre il simbolo ", occorre trovare uno dei due tasti SHIFT (funzionano entrambi) e tenerlo abbassato mentre si batte il tasto contrassegnato " 2

   Ricordarsi di sollevare il tasto SHIFT non appena (ma non prima) il segno delle " compare sullo schermo. Si è ottenuto il messaggio correntemente. Alcuni degli errori più comuni che si devono evitare sono:

   Battere con il tasto SHIFT LOCK abbassato. Si ottiene uno strano profilo con righe, cuori e picche ma non succede nulla.
   Usare due singoli ' ' invece delle ".
   Il **64** risponderà

   ? SYNTAX ERROR
   READY

   ed è possibile tentare il comando di nuovo sulla riga successiva.

Battere uno spazio tra " e T, TEST e CARD o D e ".
Battere le lettere RETURN invece di usare il tasto RETURN.
Non succederà nulla.

Usare la cifra Ø invece della lettera O posta nella parola LOAD.

Se si fa un errore, è pur sempre possibile cancellarlo battendo il tasto INST DEL. Ciascuna manovra di questo tasto cancella un carattere e sposta il cursore indietro di una posizione.

7. Se si ottiene il messaggio scritto correttamente (oppure se si è compiuto un errore nello scrivere la parola TESTCARD) la macchina risponderà

PRESS PLAY ON TAPE
(premere PLAY sul registratore)

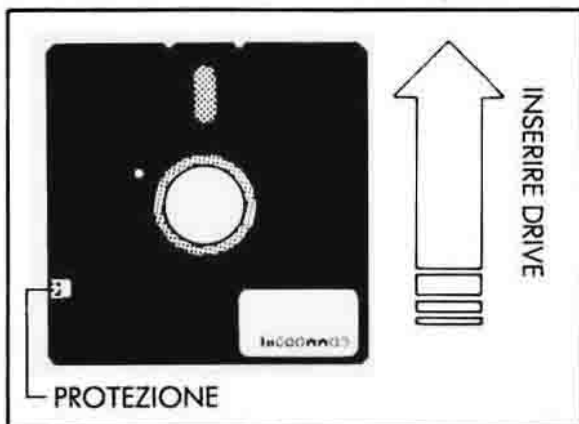
dando un immagine di questo genere:

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM 38911 BASIC BYTES FREE
READY
LOAD "TESTCARD"
PRESS PLAY ON TAPE
```

Premere PLAY sull'unità a cassetta. Attendere circa un minuto perchè il programma venga caricato.
Se il nastro continua a girare e se lo schermo mostra parecchi messaggi tipo
FOUND TESTCARD
FOUND HANGMAN
probabilmente si è scritto il nome TESTCARD in maniera scorretta. Fermare il computer premendo RUN STOP e tornare alla fase 1.
Se il nastro continua a girare e non succede nulla, assicurarsi di non aver montato un nastro vuoto. In caso contrario, c'è il sospetto che l'unità a cassetta sia difettosa per cui occorre riportarla insieme al **64** al concessionario per un controllo.
Quando il programma è finalmente caricato, la macchina dirà
READY.
Iniziare il programma battendo
RUN RETURN (4 manovre sui tasti).

Esperimento 1.1 completato	

(Per coloro che dispongono di unità a dischetti)
Importante: Leggere a fondo queste istruzioni prima di eseguirle.

1. Assicurarsi che l'unità a dischetti sia collegata al COMMODORE 64 ed alla presa di corrente. Non introdurre per il momento il dischetto nell'unità!
2. Accendere l'unità a dischetti agendo sull'interruttore posto nella parte posteriore. Si illumineranno le spie rossa e verde nella parte anteriore. Per circa un secondo l'unità eseguirà un'autodiagnosi per accertare il regolare funzionamento. Se tutto va bene la spia rossa si spegne lasciando accesa solo quella verde. Se la spia rossa non si spegne o inizia a lampeggiare, può darsi che l'unità a dischetti abbia dei problemi. In questo caso provare a spegnerla e a riaccenderla. Se il problema non scompare, riportarla al rivenditore per la riparazione.
3. Aprire l'unità a dischetti spingendo la leva verso l'interno e verso l'alto (vedere schema).

Inserire delicatamente a fondo il disco programma. L'etichetta bianca COMMODORE dovrebbe essere rivolta verso l'alto e verso destra. Premere sulla leva fino a che non scatta in avanti. Il disco è ora caricato.

4. Sulla tastiera battere il seguente messaggio:

OPEN 1,8,15,"I" RETURN

Ciò richiede in tutto 15 operazioni sui tasti, comprese le virgolette, che contano per una. Per produrre il simbolo " occorre individuare uno dei due tasti Shift (non importa quale) e tenerlo abbassato mentre si preme il tasto 2.

Ricordarsi di rilasciare il tasto SHIFT dopo aver battuto il segno " (ma non prima!).
Occore battere il messaggio nella maniera corretta. In particolare:

- Il primo carattere è la lettera O (e non la cifra 0). La O si trova nella seconda fila di tasti, tra la I e la P.
- Il quinto carattere è la cifra 1 (e non la lettera l). L'1 si trova alla sinistra della fila superiore di tasti.
- Anche il nono carattere è la cifra 1.
- Il tredicesimo carattere è la lettera I. Il tasto corretto da usare è quello che si trova tra la U e la O nella seconda fila.
- Se si batte un tasto sbagliato, usare il tasto INST DEL per cancellare ciò che si è scritto. Assicurarsi che tutto sia corretto prima di battere RETURN
- RETURN corrisponde al grande tasto posto a destra. Non battere le lettere RETURN.

Se si batte qualcosa di scorretto, l'unità a dischetti emette dei suoni strani e si accende brevemente la spia rossa. Sullo schermo compare la scritta READY. Se si commette un errore, potrebbero verificarsi parecchie cose:

(a) Lo schermo visualizza il messaggio
 ? SYNTAX ERROR
 In questo caso basta ribattere il messaggio e riprovare.
(b) Non succede nulla. Ciò probabilmente in quanto si è iniziato il messaggio con lo zero (0) anziché con la lettera O. Ribattere il messaggio.
(c) La spia rossa si accende e rimane accesa. Ciò potrebbe verificarsi per due motivi: il messaggio è scorretto (si è battuto "l" invece di "1", oppure il dischetto è stato caricato scorrettamente. In questo caso estrarre il disco e spegnere sia il computer che l'unità a dischetti, quindi ripetere dall'inizio il processo di caricamento.
(d) Se si sbaglia a battere i numeri 1, 8 o 15 la spia lampeggia e compare il messaggio ?DEVICE NOT PRESENT ERROR. In questo caso estrarre il disco, spegnere tutto e ripartire da capo.

Si supponga di aver ricevuto la risposta corretta al messaggio.
La fase successiva consiste nel battere:

LOAD"TESTCARD",8 RETURN

Ciò richiede 17 operazioni sui tasti in tutto. Anche in questo caso il messaggio deve essere assolutamente corretto, per cui occorre controllarlo attentamente prima di battere RETURN
La macchina risponde al messaggio ricercando nel disco il programma denominato TESTCARD e trasmettendolo al **64.** Essa dice inoltre cosa sta facendo e pertanto lo schermo si presenta come segue:

```
• • • • COMMODORE 64 BASIC V2 • • • •
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY
OPEN 1,8,15,"I"
READY.
LOAD "TESTCARD,"8
SEARCHING FOR TESTCARD
LOADING
READY.
```

La spia rossa dovrebbe essere spenta.
Se questo procedimento non funziona, ci sono tre probabili motivi:

(a) È stato caricato il dischetto sbagliato. La risposta sullo schermo sarà
 SEARCHING FOR TESTCARD
 ? FILE NOT FOUND ERROR
 READY.
 e la spia rossa inizierà a lampeggiare. Estrarre il disco, spegnere tutto e ricominciare da capo.
(b) TESTCARD è stato scritto scorrettamente (probabilmente è stato inserito un spazio tra TEST e CARD). La spia rossa lampeggia e sullo schermo compare
 SEARCHING FOR TESTCARD
 ?FILE NON FOUND ERROR
 READY
 Ribattere in questo caso il messaggio e riprovare:
 LOAD"TEXTCARD",8 RETURN
(c) La parola LOAD è stata scritta scorrettamente (o sono state omesse le virgolette). La macchina risponde con
 ?SYNTAX ERROR oppure con
 ?TYPE MISMATCH ERROR

Battere il messaggio correttamente e riprovare.
Una volta abituati ad eseguire il processo di caricamento, lo si può eseguire in pochi secondi. In caso di difficoltà, osservare lo schermo per stabilire dov'è l'errore. Gli errori comuni sono:

- Scambio della lettera O con lo 0 o della cifra 1 con la lettera I
- Uso di due singoli accenti (') invece di "
- Inserimento di spazi in più nelle parole tipo LOAD, OPEN e TESTCARD.

Prima di procedere, occorre richiamare l'attenzione su alcuni punti importanti relativi ai dischi:

- Non accendere o spegnere un'unità a dischetti contenente un disco. Caricare sempre *dopo* aver acceso e scaricare sempre *prima* di spegnere.
- Manipolare i dischetti il meno possibile; tenerli lontani dalla polvere, dal calore, dal freddo o dall'umidità e non toccare mai la superficie del disco esposta nella cava dell'involucro.

Quando il programma è caricato, iniziare battendo

RUN RETURN (4 manovre sui tasti).

Esperimento 1.2 completato	

(Chi ha il sistema con cassetta deve cominciare da questo punto).
Il primo programma mostra la gamma di colori che il **64** può generare e aiuta ad effettuare la regolazione fine del televisore.
Aumentare il volume e regolare la sintonia di canale lentamente fino a che non si ode la musica suonata chiaramente col minimo di rumore di fondo possibile. (È possibile riconoscere il motivo che è il Cancan di Offenbach dell'opera "Orfeo all'inferno"). Regolare quindi i comandi di luminosità e di contrasto in modo che i colori corrispondano ai rispettivi nomi e si presentino correttamente se osservati ad una ragionevole distanza. Si noterà che compare un'immagine su un bordo celeste. È possibile cambiare il bordo o margine in qualsiasi altro colore tenendo abbassato il tasto CTRL e battendo uno degli 8 tasti dei colori nella fila superiore della tastiera del **64.** Questi sono contrassegnati:

BLK	WHT	RED	CYN
nero	bianco	rosso	celeste
PUR	GRN	BLU	YEL
porpora	verde	blu	giallo

Eventualmente è possibile interrompere il programma TESTCARD premendo il tasto RUN STOP.
Quando si preme questo tasto (oppure ogni qualvolta un programma si interrompe per qualsiasi motivo), lo schermo presenta un messaggio tipo

BREAK IN 560

READY.

Il 560 nell'esempio, potrebbe essere qualsiasi numero. BREAK non significa che il **64** è rotto ma semplicemente che c'è stata un'interruzione nella sequenza di comandi che costituiscono il programma.
Il READY segnala che il **64** è pronto ad obbedire ad un altro comando della tastiera.

Se il suono continua, tenere abbassato RUN STOP e battere RESTORE. Ciò renderà silenzioso il **64** e cancellerà lo schermo.
Talvolta, quando il programma TESTCARD viene interrotto nel mezzo di un motivo, il **64** continua a suonare la nota che è stata eseguita per ultima.
È possibile interrompere l'esecuzione della nota tenendo abbassato il tasto RUN STOP e premendo RESTORE che è il tasto vicino all'angolo superiore destro della tastiera.
È comodo usare il programma TESTCARD ogniqualvolta occorre rifare la regolazione dell'apparecchio.

Esperimento 1.3 completato	

ESPERIMENTO 1·4

L'esperimento 1.4 è un gioco di indovinelli inteso a far prendere familiarità con la tastiera. Caricare il programma dal nastro o dal disco battendo

LOAD "HANGMAN" RETURN (per il nastro)
o
OPEN 1,8,15,"I" RETURN (per il disco)
LOAD "HANGMAN",8 RETURN
Lo schermo si presenterà così

```
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY
LOAD "HANGMAN"
PRESS PLAY ON TAPE
```

(per il nastro)

```
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY
OPEN 1,8,15,"I"
READY
LOAD "HANGMAN,"8
```

(per il disco)

Quando il programma è caricato e compare il messaggio READY., battere

RUN RETURN

per dare inizio al gioco. Se non si sa come giocare, basta provare qualche lettera e osservare (e ascoltare) ciò che succede. Si afferrerà rapidamente il concetto del gioco.
Proseguire il gioco a piacere e usare questa possibilità per prendere familiarità con le lettere disposte sulla tastiera.

Esperimento 1.4 completato	

Note per gli utilizzatori dei dischi:
Se avete già fatto girare un programma (come TESTCARD) e non avete spento la macchina, potete omettere il comando OPEN.

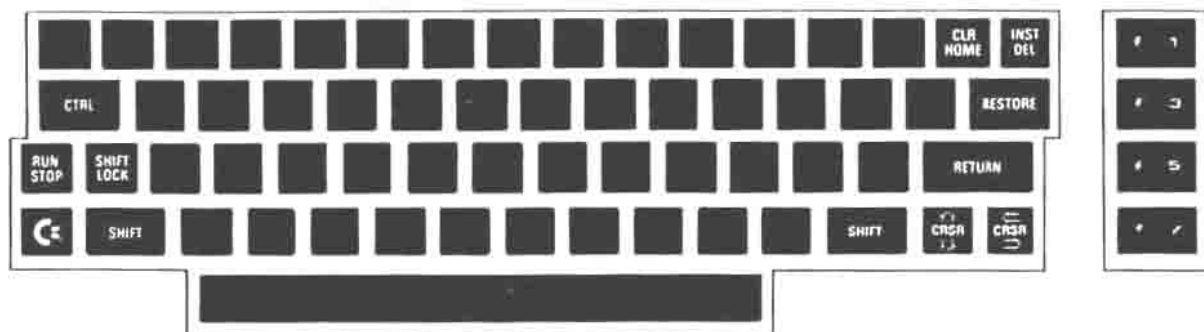
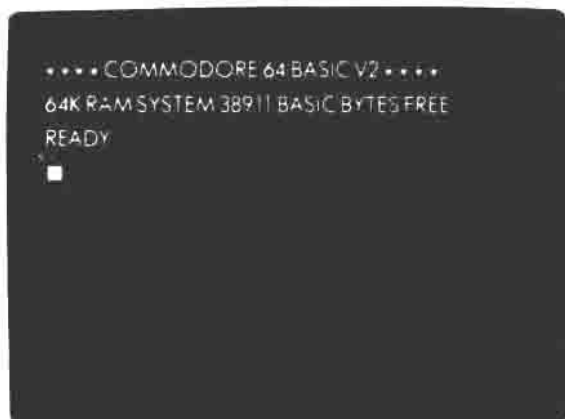
UNITA':2

Bentornati. Questa unità riguarda la tastiera del **64** e spiega come usarla per scrivere messaggi e disegnare immagini sullo schermo.
Se si è usato in precedenza una normale macchina da scrivere, la tastiera del computer apparirà familiare. Si trovano le lettere, i numeri e la maggior parte dei segni nei posti abituali e ci saranno i soliti tasti delle maiuscole (shift e relativo blocco delle maiuscole) — sebbene funzionino in modo leggermente diverso sul **64.**
Per contro non occorre scoraggiarsi se non si è mai scritto a macchina in precedenza. Occorrerà in questo caso un po' più di tempo per abituarsi alla tastiera del **64,** ma questa è la sola differenza.
Solo per questa unità, non usare il tasto RETURN a meno che non si dica di farlo. Come si è visto nella unità 1, questo tasto è quello che fa sì che la macchina esegua qualche cosa ad esempio il caricamento di programmi o la somma di qualche numero. Attualmente, basterà usare lo schermo per cui non occorre l'aiuto del computer. Se si preme RETURN, il **64** tenterà soltanto di ubbidire all'immagine o al messaggio appena battuti interpretandolo in maniera errata e guastandone l'aspetto.
Un altro simbolo che si dovrebbe evitare di usare in questo caso è quello delle virgolette ("). Questo segno ha un significato speciale e altera il modo in cui lo schermo reagisce a molti degli altri tasti sulla tastiera. Se sullo schermo compare un segno di virgolette, può essere più difficile disegnare immagini utili. Si imparerà altro su questo carattere più avanti. Per ora è bene lasciarlo stare!
Si può trovare questa lista di "cose da non fare" abbastanza allarmanti. Eccone un'altra: non preoccuparsi! A differenza dei computer della fantascienza, il **64** non ha un comando di autodistruzione. È assolutamente impossibile danneggiare la macchina battendo sulla tastiera. Alcuni profili di caratteri che contengono "o RETURN faranno sì che esso si comporti stranamente e alcune sequenze, che si potrebbero battere inavvertitamente in un momento di disattenzione, faranno sì che il computer cessi di reagire ai comandi. Questi problemi sono solo provvisori: è *sempre* possibile risolverli spegnendo il computer per 30 secondi e quindi riaccendendolo.

I 66 tasti sulla tastiera sono divisi in due gruppi:
- *50 tasti dei simboli,* che consentono al **64** di disegnare caratteri sullo schermo.
- *16 tasti di funzione* che controllano il modo in cui i caratteri sono disegnati.

I tasti di funzione sono:

I dieci tasti dei simboli, contrassegnati da 1 a 0, hanno a loro volta talune funzioni di controllo.
Confrontare la tastiera con il grafico che segue e identificare i vari tasti di controllo.

Premere parecchie volte SHIFT LOCK e notare che esso ha due posizioni — sollevato e abbassato. Infine, assicurarsi che si trovi nella posizione sollevata.

Avviare ora la macchina nel modo normale. Immediatamente al di sotto del messaggio READY. si vedrà comparire il cursore lampeggiante ■

In questo esperimento si esaminerà come si muove il cursore quando i simboli vengono disegnati sullo schermo. Lo scopo del cursore è dimostrare dove verrà a disporsi il successivo carattere battuto. Battere alcune lettere e osservare il cursore spostarsi sullo schermo. Notare che ogni carattere *sostituisce* il cursore, che si sposta quindi alla successiva posizione.
Riempire un'intera riga con lettere fino a che il cursore non si trova all'estrema destra dell'area bianca. Battere un'altra lettera e osservare cosa succede: il cursore salta automaticamente all'inizio della riga successiva.
Prima di procedere, contare il numero di lettere sullo schermo e inserirlo nella casella:

Ci sono [ ] spazi per i caratteri in ciascuna riga sullo schermo.
Successivamente, battere altre righe e continuare fino a che non si raggiunge la riga inferiore dello schermo. Contare il numero di righe che compaiono e scrivere il numero nel riquadro che segue. Ricordarsi di comprendere le righe vuote al di sopra e al di sotto del messaggio:

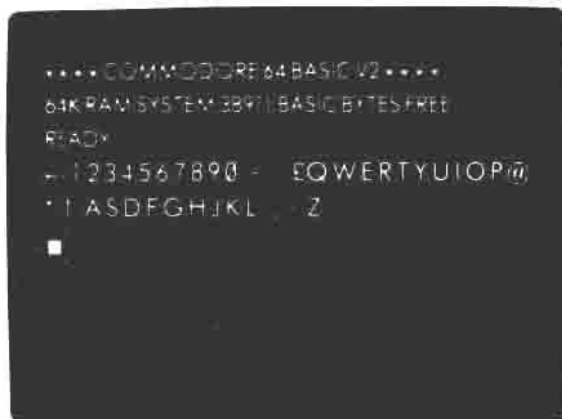
38911 BYTES FREE

In uno schermo pieno di caratteri ci sono [ ] righe.

Ora riempire l'ultima riga fino a che il cursore non raggiunge l'angolo inferiore destro dello schermo. Battere un altro carattere e osservare il cursore. L'intero schermo si muove verso l'alto e il cursore si muove all'inizio della successiva riga vuota che compare in basso. Ogni riga vuota è ottenuta a spese di una corrispondente riga nella parte superiore dello schermo, che scompare. Le righe superiori sono sparite e non c'è modo di riportarle indietro a meno che non se ne siano fatte delle copie in precedenza.
Riempire altre righe per avere la certezza che il sistema crei sempre spazio in fondo allo schermo per altro testo.

Esperimento 2.1 completato	

Il **64** ha 50 tasti per i simboli ma ne può visualizzare un numero molto più grande, fra cui lettere, numeri, segni di punteggiatura e simboli matematici ed una vasta gamma di caratteri semigrafici che possono essere combinati per creare immagini diverse. Tutti questi diversi caratteri, possono essere scelti usando uno o l'altro dei due tasti SHIFT (sono collegati all'interno del computer) e il tasto speciale Commodore contrassegnato C=
Rinizializzare il sistema e scrivere la riga seguente:
← 1 2 3 4 5 6 7 9 0 + − £ Q W E R T Y U I O P @
* ↑ A S D F G H J K L : ; = Z
(Trattasi di tutti i tasti dei simboli nelle 3 file superiori mentre la Z è in quella inferiore. Fare attenzione a non toccare i tasti di funzione mentre si batte).

```
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY
←1234567890 -  £QWERTYUIOP@
* ↑ ASDFGHJKL    Z
■
```

Ora tenere abbassato uno dei tasti SHIFT e battere di nuovo la riga. Si otterrà una riga pressochè completamente diversa di simboli (compreso il simbolo ", ma questo non deve preoccupare se si seguono le istruzioni). Copiare i simboli nella seconda fila della tabella che segue e notare come alcuni dei segni grafici (ad esempio quelli su U e I) si uniscono. Se l'immagine sul televisore è leggermente confusa può essere di aiuto osservare i segni disegnati sui tasti stessi.
Notare come i simboli grafici solitamente raggiungono il bordo dei piccoli quadrati che essi occupano in modo da potersi toccare l'uno con l'altro.

SYMBOL	←	1	2	3	4	5	6	7	8	9	Ø	+	-	£	Q	W	E	R	T	Y	U	I	O	P	@	*	↑	A	S	D	F	G	H	J	K	L	:	;	=	Z
SHIFT																																								
C=																																								

SYMBOL	X	C	V	B	N	M	,	.	/																															
SHIFT																																								
C=																																								

Battere ora la riga una terza volta ma stavolta tenendo abbassato il tasto [C=]. Molti dei segni sono di nuovo diversi. Copiare la riga della terza fila della tabella.
Per esaminare gli altri segni grafici, ripetere l'esperimento con la riga

X C V B N M , . /

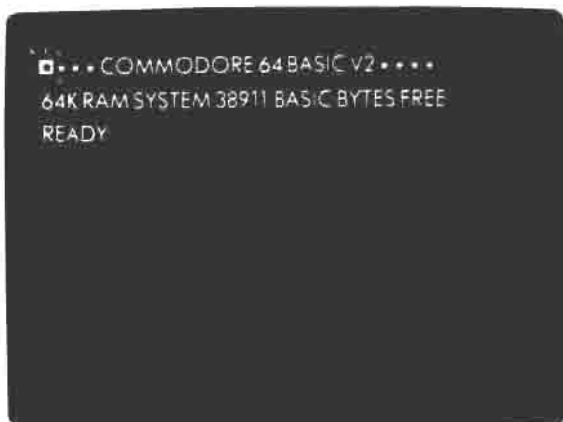
seguita da 31 spazi che portano al margine dello schermo.
Riempire l'ultima riga con spazi.
Notare e ricordare che la cifra Ø è diversa dalla lettera O. Occorre sempre usare lo Ø per fare riferimento ad un numero e non la lettera.

Premere contemporaneamente [C=] e [SHIFT]. Molte delle maiuscole sullo schermo cambieranno in minuscole. Premere di nuovo insieme i tasti e ritorneranno le maiuscole. In generale, è possibile usare un'intera serie di caratteri grafici o una serie limitate di lettere minuscole ma non entrambe contemporaneamente. L'uso delle lettere minuscole sarà spiegato nel secondo volume di questo corso.

Esperimento 2.2 completato	

ESPERIMENTO 2·3

Finora ci si è limitati a visualizzare caratteri strettamente in sequenza, da sinistra a destra e dall'alto verso il basso. Questo è un modo noioso per disegnare un'immagine. Sarebbe molto più comodo se si potesse disporre di simboli di testo e grafici in qualsiasi posizione voluta.
Ciò può essere fatto con i tasti di controllo del cursore — ce ne sono tre, e cioè [CLR HOME], [CRSR ⇕], e [CRSR ⇔].
Quando si batte il tasto [CLR HOME] si sposta il cursore riportandolo in posizione di partenza e cioè nell'angolo superiore sinistro dello schermo. Riavviare la macchina e battere questo tasto. Si vedrà il cursore spostarsi sull'★ nell'angolo superiore sinistro dello schermo. L'★ rimane visibile in quanto il cursore è trasparente; ma se si batte un altro carattere (o uno spazio) il simbolo sotto il

■ ★ ★ ★ COMMODORE 64 BASIC V2 ★ ★ ★ ★
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY

cursore viene sostituito da uno nuovo. Provare a battere un = invece dell' ★. Battere = tre altre volte in modo da avere

= = = = COMMODORE 64 BASIC V2 ★ ★ ★ ★
come riga superiore dello schermo.
Si può ora variare ★ ★ ★ ★ sulla destra in = = = =, in modo da mantenere la riga simmetrica. Se si sposta il cursore battendo gli spazi, si cancellerà il titolo al centro della riga. Il modo corretto consiste nell'usare il tasto [CRSR ⇔]. Ogni volta che si batte questo tasto, il cursore si sposta di uno spazio a destra ma *senza* cancellare nulla durante lo spostamento.
Provare a spostare il cursore sul primo ★ a destra e quindi inserire quattro serie di =. La riga superiore diventa
= = = = COMMODORE 64 BASIC V2 = = = =
e il cursore si sposta alla sinistra della riga successiva.

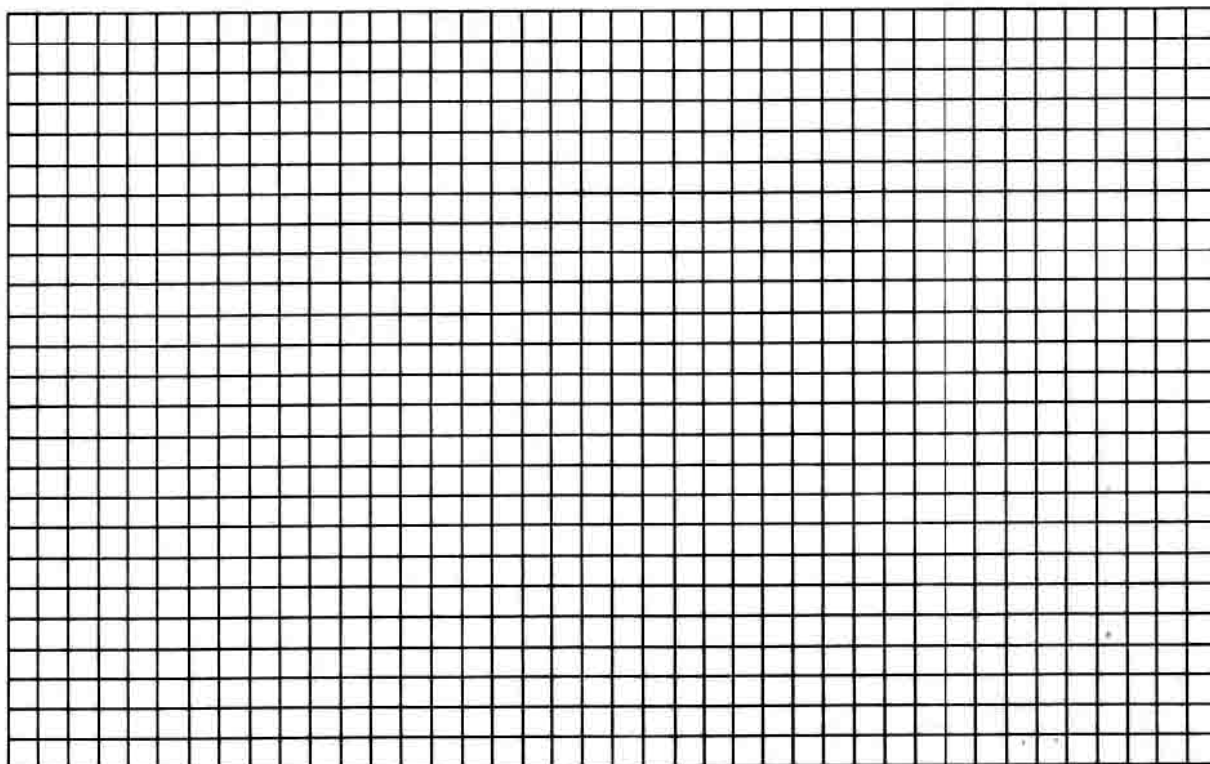
Se si tiene abbassato in continuazione il tasto [CRSR ⇔] dopo una pausa, il cursore si sposta automaticamente alla velocità di circa 10 posizioni al secondo, riga dopo riga. Ciò è utile per spostarsi rapidamente.
Quando viene premuto [CRSR ⇔] contemporaneamente a [SHIFT], il cursore si muove all'indietro. Quando raggiunge l'inizio di una riga si muove verso l'alto e ritorna alla fine della riga precedente.
Successivamente, provare a ritornare alla prima riga e a cambiare di nuovo i segni di = in ★.
Spostare il cursore in basso alla riga di fondo dello schermo e osservare cosa succede quando si arriva oltre la fine della riga: l'intero schermo si muove verso l'alto come se fosse stato aggiunto un altro carattere.
Ora ritornare al punto di partenza e provare a muovere il cursore all'indietro. Lo schermo non si muove verso il basso come ci si potrebbe aspettare; non succede nulla e il cursore rimane nella stessa posizione.
I tasto [CRSR ⇕] sposta il cursore verso l'alto e verso il basso di un intera riga alla volta. Fare qualche esperimento con questo tasto e assicurarsi di aver compreso come funziona.
Successivamente riempire lo schermo con alcuni caratteri e segni grafici quindi premere [CLR HOME] tenendo abbassato il tasto SHIFT. Il cursore si sposta al punto di partenza e lo schermo viene cancellato, fornendo una superficie intatta sulla quale lavorare.
Compilare la seguente tabella:

Tasto	Effetto	
	Senza shift	Con shift
[CLR HOME]	Sposta cursore al punto di partenza	
[CRSR ⇔]		Sposta il cursore di uno spazio all'indietro
[CRSR ⇕]		

Fare ora pratica creando disegni sullo schermo usando i simboli grafici o i tasti di comando del cursore. Iniziare con alcuni semplici profili geometrici tipo quadrati, rombi, triangoli e piccoli cerchi. Se si fa un errore, spostare il cursore indietro e battere il carattere corretto. La barra di spazio libererà dai caratteri disposti nel posto errato.
Una volta presa familiarità con la creazione di segni grafici, disegnare un riquadro tipo questo contenente il proprio nome.

MARIO
ROSSI

Ora disegnare qualche carta da gioco con angoli curvi e i simboli corretti (suggeriamo di tenere le carte nere con un valore di 10 o minore). Infine, per chi è dotato di talento artistico, provare a battere qualcosa tipo un animale, una nave spaziale o un volto umano.

Pianificare per prima cosa l'immagine usando la griglia che segue.

Esperimento 2.3 completato

ESPERIMENTO 2·4

Tutti compiono errori durante la battitura. Se capita di avere una lettera sbagliata nel centro di una parola, è possibile correggerla con il controllo del cursore. Per esempio se si batte AUSTRAPIA invece di AUSTRALIA, si può spostare il cursore sulla lettera P e cambiarla in L. Provare!
Sfortunatamente, se come risultato si ottiene il numero sbagliato di lettere (troppo poche o troppo numerose), questo metodo non funziona.

Un modo più potente è fornito dal tasto INST DEL, che consente di inserire o rimuovere caratteri dallo schermo.

Quando si batte INST DEL da solo, cancella il carattere alla sinistra del cursore e sposta tutti gli altri caratteri di uno spazio verso sinistra per riempire lo spazio vuoto.
Per esempio, si supponga di aver battuto erroneamente INXDIA invece di INDIA. Ci si vuole naturalmente liberare dalla X per cui occorre disporre il cursore sopra la D e battere INST DEL
La X scompare e DIA si muove a sinistra, lasciando INDIA (senza spazio intermedio).

Provare ora a usare il tasto INST DEL per eseguire alcune correzioni ad esempio:

CHAINA in CHINA
EEGYPT in EGYPT
FINLANDIA in FINLAND
AUSTRALIA in AUSTRIA

In pratica, l'uso più comune del tasto INST DEL è quello di liberarsi del carattere o dei caratteri appena battuti. Il tasto rimuoverà l'ultimo simbolo e riposizionerà il cursore, il tutto in un solo movimento. Ci si abituerà presto a battere INST DEL ogni qualvolta si compie un errore di battitura.

L'altra funzione del tasto INST DEL può essere richiamata battendolo come carattere preceduto da SHIFT: cioè tenendo abbassato il tasto battere INST DEL.

Questa funzione è usata per *inserire* spazi al centro di parole o di righe. Questi spazi possono essere riempiti con caratteri nel modo solito.
Provare l'esempio seguente cambiando AUSTRIA in AUSTRALIA.

Cancellare lo schermo (e) e battere

AUSTRIA

Spostare il cursore di nuovo sulla I.

Tenendo abbassato il tasto shift, premere INST DEL due volte. Ogni volta la IA si sposta di uno spazio a destra. Il cursore rimane nello stesso punto cosicchè dopo due mosse si ottiene

Terminare ora inserendo AL. Spostare il cursore oltre la fine della parola.

Per far pratica nell'uso del tasto INST DEL, cancellare lo schermo, riempirlo con la lista di parole sulla sinistra e quindi cambiare ciascuna parola nella corrispondente sulla destra:

HOTEL	MOTEL
MICROPHONE	MICROCOMPUTER
PLYWOOD	WOOD
ANGLE	ANGEL
CHAP	CHEAP
WRITEN	WRITTEN
ACTOR	AUTHOR
BALL	BARREL
WIRE	REWIRE
FLOWER	FLOUR
MOON	MORON
PIDGIN	PIGEON
TACT	TACIT
HORSE	HOARSE
WING	WARRING
TAXI	TAX
RED	READY
MERRY	MERCURY
JOVE	JUPITER
PAL	PASCAL
BACK	BASIC
JAVA	JAMAICA

Ed ora ricambiali di nuovo per riportarli alla versione originale.

Esperimento 2.4 completato	

Il programma dell'unità 2 è intitolato SPEEDTYPE. Esso aiuta a prendere familiarità con la tastiera. Caricarlo dalla cassetta battendo
LOAD "SPEEDTYPE"
oppure da disco battendo
OPEN 1,8,15,"1"
LOAD "SPEEDTYPE",8
e avviarlo con il comando RUN facendo pratica nel suo uso fin quando lo si ritiene necessario.

UNITA':3

Il **64** è un computer previsto per l'uso dei colori. Questa unità presenta alcuni dei modi coi quali è possibile far sì che la macchina disegni molte immagini colorate sullo schermo del televisore.
Se il televisore è in bianco e nero, ovviamente non ci si devono aspettare risultati brillanti dall'Unità 3. In ogni caso bisogna lavorare allo stesso modo.

ESPERIMENTO 3·1

Usare il programma TESTCARD (unità 1) per assicurarsi che il ricevitore TV sia correttamente regolato. Interrompere il programma premendo RUN STOP e RESTORE

Si vedrà che il cursore in questa fase è blu. Ora il cursore può cambiare colore e oltre a dire dove verrà disposto il prossimo simbolo, dice anche di quale colore sarà. Provare a battere una sequenza di simboli ▬ (tasti C= e I) e notare che essi compaiono in blu che è l'attuale colore del cursore.
(In alta risoluzione, alcune volte, può capitare che il carattere non abbia un colore ben definito. Ciò è la conseguenza dell'uso di un normale televisore che ha un'ampiezza di banda a radiofrequenza molto stretta. La difficoltà può essere risolta da un monitor a colori, più costoso ma raramente ne vale la pena a meno che si abbia in programma di presentare una grande quantità di testi in diversi colori).

Figura 1

Il colore del cursore può essere cambiato in qualsiasi momento battendo uno degli 8 tasti colore mentre si tiene abbassato il tasto CTRL
I tasti del colore sono contrassegnati da 1 a 8 e portano anche abbreviazioni dei rispettivi colori che essi controllano.

Tenere abbassato CTRL e battere i tasti dei colori in successione.
Quando si batte 1 (BLK) il cursore diventa nero.
Quando si batte 2 (WHT), il cursore diventa bianco.
Gli altri tasti cambiano il colore del cursore rispettivamente in rosso, celeste, porpora, verde, blu e giallo. Quando il cursore è blu, risulta invisibile sullo sfondo blu e infatti sembra scomparire.
È possibile ottenere un'altra serie di 8 colori usando il tasto C= invece di CTRL. Stavolta si tratta di tonalità pastello meno brillanti.

C= e BLK dà arancio
C= e WHT dà marrone
C= e RED dà rosso chiaro
C= e CYN dà grigio scuro
C= e PUR dà grigio medio
C= e GRN dà verde chiaro
C= e BLU dà blu chiaro
C= e YEL dà grigio chiaro - il colore "solito" del cursore.

Provare a creare ora qualche immagine a colori. Un buon modo per cominciare consiste nel creare qualche barra colorata di varia lunghezza. Usare il segno grafico ▀ (C= e U) per costruire ciascuna barra. Per esempio, per creare una barra rossa lunga 5 simboli, tenere abbassato CTRL e battere 3 (RED); ciò cambia il cursore in rosso.

Tenere quindi abbassato C= e battere U cinque volte.
Una volta fatta l'abitudine ai tasti dei colori, provare a disegnare una "tabella dei risultati delle elezioni" come indicato in Figura 1. Creare le lettere in nero per evidenziare il colore delle barre.

Esperimento 3.1 completato	

ESPERIMENTO 3·2

Oltre ad occuparsi dei colori dei singoli simboli, il **64** può controllare i colori del riquadro esterno e del fondo sul quale i simboli appaiono. Il video non ha qualsiasi tasto specializzato per controllare questi colori e il solo modo per cambiarli è di impartirgli un comando speciale.
Battere POKE 53280,0. Controllare accuratamente e correggere se necessario e quindi battere il tasto RETURN. Il margine sullo schermo del televisore diventa immediatamente nero.
Questo comando speciale è costituito da tre parti:

POKE: Si tratta di una cosiddetta *parola chiave*.
53280: Si tratta del cosiddetto *indirizzo* che identifica la parte del **64** che si preoccupa dei colori del margine e del fondo. Qualsiasi speciale comando per cambiare uno di questi due colori, deve *sempre* fare riferimento a questo indirizzo.
0 : Questo è un codice che indica "nero".

Per scegliere un altro colore del riquadro, basta usare il codice appropriato. La tabella dei colori è la seguente:

Colore	Nero	Bianco	Rosso	Azzurro
Codice	0	1	2	3
Colore	Porpora	Verde	Blu	Giallo
Codice	4	5	6	7
Colore	Arancio	Marrone	Rosso	Grigio sc.ro
Codice	8	9	10	11
Colore	Grigio med.	Verde	Blu chiaro	Grigio ch.ro
Codice	12	13	15	15

Se si desidera un riquadro giallo, battere

POKE 53280,7 RETURN

Analogamente,

POKE 53280,11 RETURN

dà un riquadro grigio scuro.

Vale la pena di notare che il numero di codie per ciascuno dei colori più brillanti è inferiore di uno al numero del testo usato per scegliere quel colore per il cursore. (Ad esempio, il tasto PUR — porpora — è contrassegnato "5" e il codice per il color porpora è 4). I codici dei colori pastello hanno ciascuno un valore di sette unità superiore a quello del tasto.
Il colore del fondo dei simboli viene controllato esattamente allo stesso modo, usando l'indirizzo 53281. Per ottenere uno sfondo grigio chiaro con un riquadro color porpora occorre battere:

POKE 53280,4 RETURN

POKE 53281,11 RETURN

Scegliere alcune combinazioni di colori, eseguire con esse POKE con i comandi speciali e trovarne una o due gradite. Ricordarsi che se si rende il cursore invisibile e la tastiera sembra non rispondere, è sempre possibile riportare il cursore indietro premendo RUN STOP e premendo RESTORE

Esperimento 3.2 completato	

ESPERIMENTO 3·3

Si saranno notati alcuni strani vuoti nei caratteri grafici. Ad esempio abbiamo ▚ ma non ▞; abbiamo ◤ e ◥ ma non ◢ o ◣. Inoltre sembra impossibile riempire un quadrato completo con il colore e pertanto costruire grandi aree della stessa tinta.
Per questo ci viene in aiuto la funzione di *campo negativo* (reverse).
Quando un carattere viene visualizzato in negativo, i colori dei caratteri e del suo fondo vengono scambiati. Provare il seguente esperimento:

Riavviare il **64**, scegliere il porpora come colore del cursore e battere alcuni caratteri compresi ▚, ◤, ◥ e uno spazio. Tenere ora abbassato CTRL e il tasto 9 (contrassegnato anche RVS ON).

Quindi sollevare CTRL e battere altri caratteri. Questi compariranno in bianco sul fondo porpora anzichè in porpora su fondo bianco. In particolare ▚ diventa ▞, ◤ e ◥ sono cambiati in ◢ e ◣, e uno spazio compare invece come un blocchetto color porpora. Diciamo che il **64** è nel modo *negativo*.
Per riportare i seguenti caratteri al modo normale, tenere abbassato CTRL e battere il tasto dello 0 (contrassegnato RVS OFF).
Il cursore non mostra se la macchina è in modo normale o negativo. Se si usano molti simboli in negativo, è facile dimenticarsi e trovarsi in dubbio sul modo in cui comparirà il successivo carattere. Questa difficoltà può essere risolta in due modi:

(a) Battendo ed osservando il successivo carattere. Se è sbagliato, cancellarlo, cambiare il modo e riprovare.
(b) Battere il tasto RVS ON o RVS OFF, come appropriato, prima di battere il successivo simbolo. Se la macchina è già nel modo corretto, questo non farà alcuna differenza.

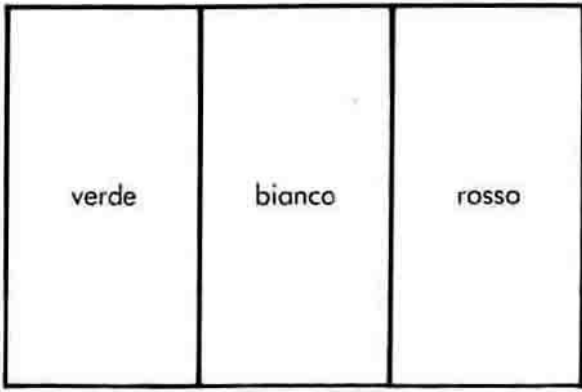
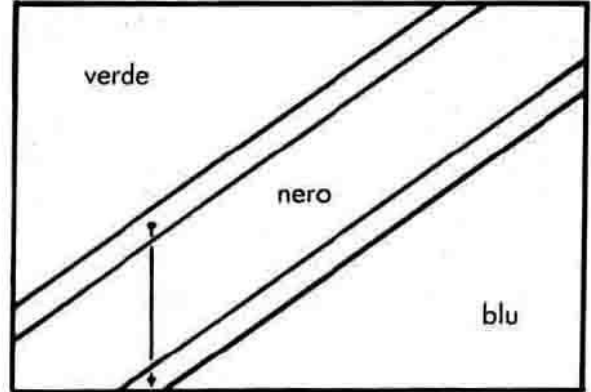
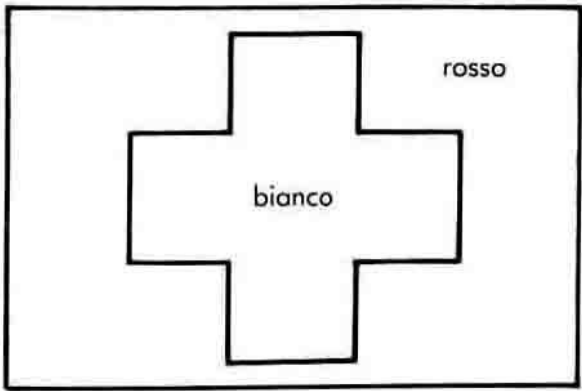
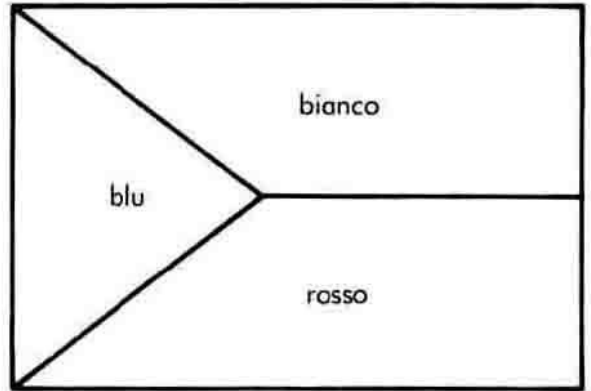
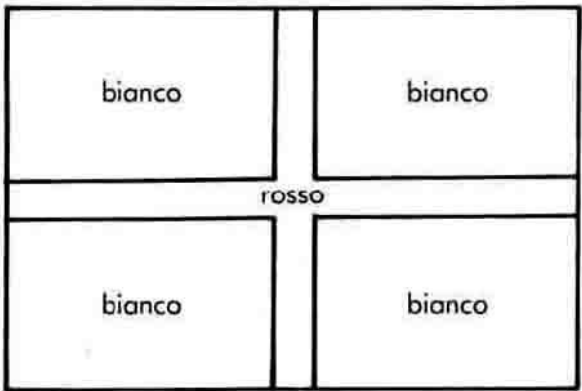
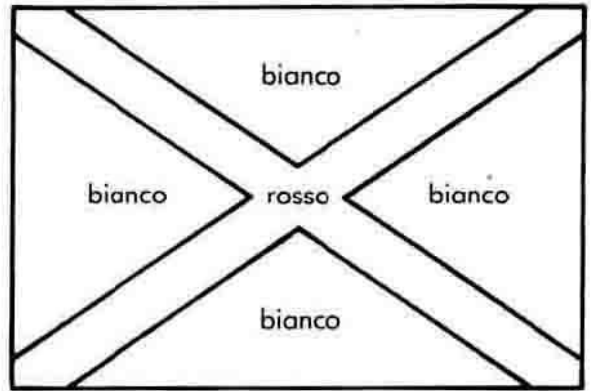
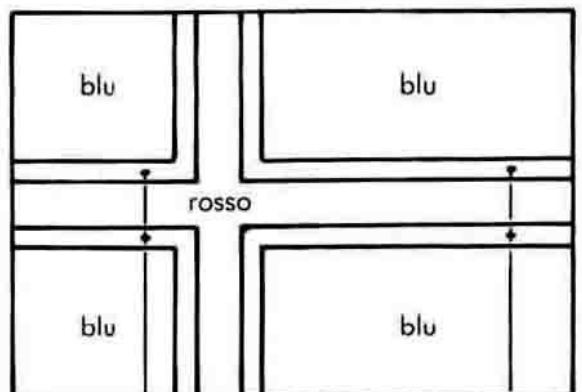
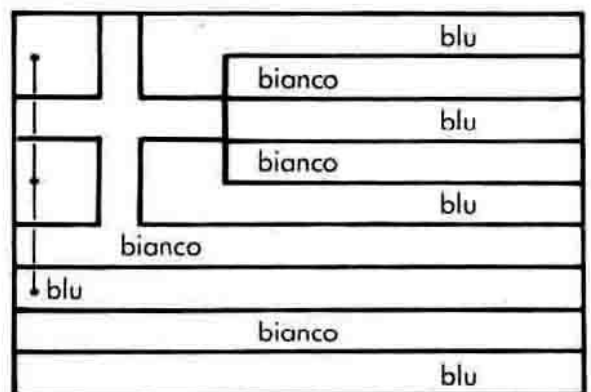
Il modo migliore per riempire lo schermo con blocchetti di un colore consiste nell'usare degli spazi in negativo. La barra di spazio è un tasto a ripetizione automatica e se lo si tiene abbassato genererà una sequenza di spazi al ritmo di circa 10 al secondo.
Il disegno delle bandiere nazionali, rappresenta un buon modo per fare pratica con l'uso dei colori. Il tipo più facile di bandiera da riprodurre è quella con strisce orizzontali, ad esempio la bandiera del Sudan.

porpora
giallo
verde

Per colorare questa bandiera, definire il margine ad un colore adatto (ad esempio nero) e il colore di fondo allo stesso colore dell'angolo inferiore sinistro della bandiera (verde). Ciò avviene col comando.

POKE 53280,0 RETURN

POKE 53281,5 RETURN

Successivamente spostare il cursore al punto di partenza, scegliere il rosso e il modo negativo (tenere abbassato CTRL e battere PUR e quindi RVS ON). Quindi tenere abbassata la barra di spazio e riempire otto righe con spazi in rosso in negativo.
Successivamente, scegliere in giallo e riempire nove righe con spazi bianchi in negativo.
Infine, cambiare il colore in verde. Ciò fa scomparire il cursore e lascia un'area verde di otto righe alla base della bandiera.
È Importante scegliere come colore di fondo uno dei colori che compaiono sull'immagine altrimenti non è possibile nascondere il cursore quando il disegno è completo. Analogamente, il colore del margine deve essere diverso da qualsiasi colore che compare nella bandiera stessa.
Una volta acquisita familiarità con le bandiere orizzontali, provare a crearne una con strisce verticali, ad esempio quella italiana. Vale inoltre la pena di disegnare quelle in cui compare una croce (Svizzera o Islanda).
Ancora più difficile sono le bandiere con elementi diagonali tipo Cecoslovacchia.
La parte della bandiera in prossimità delle linee inclinate deve essere realizzata con i segni grafici ◢ o ◣, opportunamente invertiti, se necessario. Pensandoci, si vedrà che il colore del fondo deve essere scelto in modo che si trovi su un lato di ogni riga inclinata. Nel caso della bandiera della Cecoslòvacchia, un fondo adatto è il

blu; il rosso non andrebbe bene in quanto non ci sarebbe modo per creare un elemento blu e uno giallo sulla diagonale superiore.
Se una bandiera presenta una barra diagonale, come nel caso della Tanzania, vale la pena di usare solo 25 delle 40 colonne e riempire il resto con il colore del fondo. Supponendo un fondo nero, una banda che attraversi in diagonale la bandiera della Tanzania verrebbe immessa nella forma:

Le virgole nonchè la "e" sopraindicate servono semplicemente per mostrare i diversi comandi perchè siano più facili da seguire. Non devono in effetti, essere usate quando si disegna la bandiera.

Esperimento 3.3 completato	

Disegnare la miglior immagine colorata possibile, usando l'intero schermo del **64**. È possibile facilitare il lavoro pianificando prima l'immagine, usando un foglio di carta quadrettata e matite colorate.

Esperimento 3.4 completato	

Il programma che fa parte dell'Unità 3 è un quiz, che può essere caricato battendo

LOAD "UNIT3QUIZ"

o

OPEN 1,8,15 "I"
LOAD "UNIT3QUIZ",8

UNITA':4

Nelle prime tre unità del corso ci siamo concentrati pressochè interamente sulla tastiera del **64** e sul suo uso per visualizzare un testo e disegnare immagini sullo schermo del televisore. Questa è una buona preparazione per la successiva parte del corso dove ci occuperemo di alcune delle funzioni che il **64** può eseguire su comando dell'operatore.
Come già noto, il **64** eseguirà vari lavori quando verrà comandato a farlo. I necessari comandi sono scritti in BASIC, un semplice e diffuso linguaggio per computer ideato da Kemeny e Kurtz del Dartmouth College, USA. Il BASIC ha le proprie regole di grammatica esattamente come qualsiasi altro linguaggio, ma farà piacere sapere che esse sono facili da imparare e che verranno facilmente ricordate con la pratica, senza qualsiasi sforzo speciale.
Qualsiasi comando BASIC inizia con una "parola chiave" tipo LOAD o POKE o PRINT. Ciò dice al computer quale tipo di comando si intende.
Analogamente ogni comando termina con il tasto RETURN. Questo ha due significati diversi:
se la parola chiave è la prima parola sulla riga, RETURN si può paragonare al colpo di pistola dello starter: "Ora parti ed eseguilo". Per esempio se si batte

LOAD "TESTCARD"

il caricamento effettivo inizia quando viene premuto il tasto RETURN. L'altra interpretazione di RETURN è discussa nell'unità seguente, ma eccone un breve anticipo:
se la parola chiave in un comando BASIC è preceduta da un numero, ad esempio

3 PRINT 5+7

il tasto RETURN segnala di non ubbidire al comando ma di memorizzarlo per uso successivo.
In questa unità ci concentreremo sulla prima interpretazione. I nostri comandi non saranno preceduti da numeri e RETURN sarà un segnale al **64** di intraprendere un'azione immediata.

ESPERIMENTO 4·1

Uno dei comandi più utili e flessibili è PRINT. Esso fa sì che il computer esegua qualcosa e visualizzi il risultato sullo schermo. La parola PRINT è usata in quanto il sistema originale BASIC installato a Dartmouth si basava su telescriventi che effettivamente "stampavano" le risposte su rotoli di carta.
L'esperimento 4.1 è organizzato in tre fasi. Innanzitutto proveremo un certo numero di diversi comandi PRINT e prenderemo nota accurata dei risultati. Successivamente, discuteremo le caratteristiche del comando che sono state indicate negli esempi ed infine esamineremo alcuni nuovi comandi PRINT e cercheremo di vedere che cosa il computer farà con essi. Le risposte possono essere confrontate usando il computer stesso.
Iniziamo battendo questi comandi e terminando ciascuno di essi con il tasto RETURN. Assicurarsi di impostare i comandi correttamente usando i tasti di controllo del cursore per correggere eventuali errori di battitura. Prendere nota accurata delle risposte nelle caselle all'uopo predisposte. Le prime due caselle sono già state compilate:

PRINT 999	999 READY.
PRINT "HELLO"	HELLO READY.
PRINT −56	
PRINT 3+4+4	
PRINT 5★7	

PRINT 27/7	
PRINT "64 CBM"	
PRINT COMMODORE 64	
PRINT 3,5	
PRINT 3;5	
PRINT "RABBIT", "DOG"	
PRINT "CAT"; "FISH"	
PRINT "3+5"	
PRINT 29–12; "LIONS"	
PRINT 1;2;3;4	

Prima di procedere, studiare le note accuratamente per rendersi conto di quante diverse caratteristiche di PRINT è possibile ottenere. Un aspetto comune delle risposte è che tutte sono seguite da READY. Ma ciò è vero per qualsiasi comando inviato direttamente allo schermo, cosicchè è difficile considerarlo una proprietà speciale di PRINT.
Qui ci sono i punti importanti del comando PRINT:

1. Il comando può gestire sia *numeri* che *stringhe* e lo fa in modi diversi: un numero può essere indicato esplicitamente (ad esempio 999) oppure sotto forma di un'espressione o "somma" che il computer elabora. Le espressioni nei nostri tentativi erano 3+4+5, 5★7, 27/7 e 29-12, cosicchè vediamo che il **64** può sommare, sottrarre, moltipfllicare e dividere. I segni ★ e / significano rispettivamente moltiplicazione e divisione. (Se si è interessati ad usare il computer per calcoli matematici più avanzati, farà piacere sapere che queste espressioni possono essere anche molto complicate, potendo comprendere parentesi e tutte le funzioni speciali che ci si può aspettare da un calcolatore scientifico. Si consiglia di osservare l'Appendice A che è un'unità extra intesa particolarmente per questo scopo).
   Una *stringa* è qualsiasi sequenza di caratteri racchiusa tra virgolette. Il comando PRINT ripete tale stringa esattamente come è stata indicata, senza cercare di elaborarla in alcun modo. Le stringhe del nostro testo erano "HELLO","64 CBM", "RABBIT",
   "DOG", "CAT", "FISH", "3+5" e "LIONS".
   Notare che "3+5" *non* è un'espressione quantunque possa sembrare tale; è chiuso tra virgolette per cui deve essere una stringa. Se si vuole visualizzare una stringa ma si dimentica di porre le virgolette prima e dopo, probabilmente si otterrà 0 (quantunque talvolta ciò possa significare qualche cosa).
2. Il comando PRINT può gestire due o più stringhe contemporaneamente. Se le due sono separate da virgole, il secondo risultato è ben distanziato sullo schermo (esattamente sull'altra metà). Se viene usato un punto e virgola, la separazione è minore. In particolare le stringhe non sono separate per nulla (come ad esempio nel caso di "CATFISH"). I numeri visualizzati dal computer sembrano essere separati in quanto ogni numero è sempre preceduto da uno spazio (oppure da un segno meno se è negativo) e sempre seguito da un altro spazio. Se i record non mostrano tutto ciò molto chiaramente, ripetere il comando.

   PRINT 1;2;3;4;

   e "misurare" il risultato spostando il cursore su di esso.
3. Ora guardiamo gli spazi all'interno del comando. La parola chiave PRINT deve essere compatta (cioè le lettere non devono essere separate da spazi(e qualsiasi spazio all'*interno* di un stringa a partire da quella stringa verrà riprodotto. Altrimenti gli spazi tra le stringhe o i numeri sono ignorati. Pertanto

   PRINT 3 +5;7; 8

   darebbe lo stesso risultato che

   PRINT3+5;7;8

   Questa regola — che gli spazi nei comandi sono ignorati ovunque salvo che all'interno delle parole chiave e nelle stringhe — è generalmente valida per l'intero BASIC.
4. Se si compie un errore nella parola chiave, oppure se si scrive un'espressione che non ha senso (ad esempio 5★★7), il computer respinge il comando con il commento

   ?SYNTAX ERROR

Questo è il gergo del computer per dire che sono state infrante le regole del BASIC. Non c'è nulla da fare salvo correggere il comando e provarlo di nuovo.
Procedere ora con la seguente lista di comandi PRINT cercando di prevedere che cosa farà il **64** con ciascuno di essi. Indicare come i risultati saranno distanziati; ciò è altrettanto importante che ottenere risultati esatti. Per contro, non preoccuparsi di scrivere READY. ogni volta.
Alcuni dei comandi possono contenere errori deliberati.
Controllare le risposte sul **64.** Se sono stati compiuti errori e non è possibile rendersi conto del perchè, ritornare daccapo e ripetere l'intero esperimento fino a che le idee si chiariscono. Notare che il **64** esegue sempre i calcoli che comportano moltiplicazioni e divisioni prima di qualsiasi addizione e sottrazione.

Ad esempio:

PRINT 5+2★7 darà 19

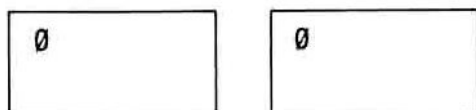
e

PRINT 5+2+6/3-3 darà 6.

Comando	*Previsione*	*Risultato del* ***64***
PRINT 94		
PRINT 8—5		
PRINT 3★2+5		
PRINT "OH DEAR"		
PRINT ENOUGH		
PRINT "1★/3"		
PRINT 3;47		
PRINT 2+2;2−2;2★2;2/2		
PRINT "CLOUD"; "BURST"		
PRINT 8—7		
PRINT "18", "MICE"		
PRINT 53+★7		
PRINT −1;−2;−3		

Esperimento 4.1 completato	

Se un computer potesse eseguire un solo comando alla volta non sarebbe particolarmente utile a nessuno. Tuttalpiù potrebbe servire quanto un calcolatore non programmabile. La maggior parte dei lavori utili del computer consiste di intere sequenze di comandi, controllate da una serie di istruzioni dette "programmi". Quando i comandi sono eseguiti in sequenza, deve esserci un modo per tener nota, ossia ricordare di quanto è preceduto il lavoro e di trasmettere i risultati di un comando al successivo. La memoria che serve per collegare i comandi è fornita sotto forma di *variabili.*
Prima di discutere le variabili BASIC, illustreremo un'analogia umana. Si supponga di fungere da segnapunti in un incontro di calcio. Le istruzioni potrebbero essere le seguenti:

Prima che la partita inizi, disegnare due riquadri, contrassegnarli con il nome delle squadre e scrivere degli zero all'interno, così:

Ogniqualvolta una delle due squadre segna un goal, sostituire il numero scritto più di recente nella corrispondente casella con gli stessi numeri più uno. Cancellare o barrare il vecchio numero. Ad un certo punto, durante la partita, la situazione potrebbe essere

Al fischio finale, usare la lavagna per visualizzare i nomi delle squadre unitamente ai numeri (più recenti) all'interno dei riquadri e cioè:

INTER	2
MILAN	5

In questo esempio le caselle o riquadri sono *le variabili.* I numeri nelle caselle servono per ricordare lo stato corrente della partita e cambiarlo di volta in volta come necessario; ma le label rimangono le stesse per la durata dell'incontro. Le istruzioni per usarle sono molto semplici, ma a prova di errore.
La memoria del **64** (probabilmente ha 38911 byte — ricordi) si può in un certo qual modo paragonare ad una lavagna. Quando la macchina viene accesa la lavagna viene completamente cancellata, quindi ogniqualvolta viene citata per la prima volta una variabile, il computer "disegna una casella" accantonando una parte della memoria e la contrassegna con il nome scelto dall'utente umano. Quindi "scrive un numero nella casella" memorizzando il valore appropriato nella memoria che è stata così accantonata. Il comando BASIC che fa sì che il computer faccia tutto ciò, contiene la parola chiave LET. Esaminiamo questo comando in dettaglio:

LET X = 5

Qui il nome della variabile è X. Il **64** creerà una casella denominata X (se non lo ha già fatto) e vi inserirà il numero 5. Se esiste già una variabile X, non viene accantonato altro spazio; il 5 semplicemente *sostituisce* il valore precedente. Studiare i casi seguenti:

	X non esiste *(Caso 1)*	X esiste *(Caso 2)*
Prima	(Memory empty)	[37] X
Dopo	[5] X	[5] X

Risultato di LET X = 5

Quando si impartisce un comando LET, il computer dice semplicemente

READY.

Non c'è traccia sullo schermo di ciò che la macchina ha fatto. Fortunatamente, siamo aiutati dal comando PRINT che visualizza il valore di una variabile ogniqualvolta è citata per nome. Provare le seguente sequenza di comandi:

	Your results
LET Z = 14 PRINT Z	
LET Z = 31 PRINT Z	

Se si mantiene l'ordine esatto, il primo valore di Z da stampare sarà 14 e il secondo 31. Il primo comando LET crea una variabile denominata Z e le attribuisce il valore 14; il secondo semplicemente ne cambia il valore in 31.
A questo punto dobbiamo dare alcune semplici regole sulle variabili e sui loro nomi.
In BASIC ci sono due tipi di variabili:

- Variabili numeriche, per memorizzare i numeri.
- Variabili stringa per memorizzare stringhe (ad esempio parole o frasi)

La scelta dei nomi per le variabili è abbastanza limitata. Una variabile numerica può essere richiamata da una singola lettera seguita da una cifra o da due lettere. Alcuni esempi di nomi possibili per le variabili numeriche sono:

A, X, B5, TX, PQ

I nomi per le variabili stringa terminano sempre col segno $. Salvo ciò le regole per le variabili stringa sono le stesse di quelle per le variabili numeriche. Gli esempi sono:

C$, Z$, P7$, DB$

Per mostrare l'uso delle variabili stringa, provare a battere

```
LET T$ = "BUON"

PRINT T$; "GIORNO"
```

Il valore che segue il segno = in un comando LET non deve necessariamente essere un semplice numero o una stringa; può essere un'*espressione* e inoltre esso può usare i valori correnti delle variabili facendo riferimento ai loro nomi. Per esempio, osservare la sequenza di comandi:

```
LET Q = 5

LET S  = Q+3
```

Il primo crea una variabile denominata Q (è una variabile numerica a causa del suo nome) e imposta il suo valore a 5. Il secondo crea una variabile denominata S. Questo prende il valore di Q, gli aggiunge 3 e mette il risultato in S. Per illustrare questo punto, provare ad eseguire questi due comandi ed esaminare il risultato battendo

```
PRINT Q; S
```


Si osservi ora la seguente sequenza e si *preveda* il risultato dell'istruzione PRINT in ciascun caso:

```
LET AA = 15
LET B = 33—AA
PRINT AA,B        [          ]

LET D = 3
LET E = D★D+7
LET F = E—D
PRINT F;E;D       [          ]

LET F=4
LET F=F+1
PRINT F           [          ]
```

L'ultima era esatta? Alcuni potrebbero trovarla un pochino complicata.
Non c'è limite al numero di diversi valori che una variabile può assumere purchè ne contenga uno alla volta. Un comando tipo

LET F = F+1

significa: Per *prima* cosa calcolare l'espressione (prendendo il valore di F e aggiungendo 1)
Quindi inserire il risultato nella casella F, sostituendo il valore precedente.

In altre parole, il comando fa sì che il **64** aggiunga 1 al valore corrente di F.
I segni che consentono di combinare i numeri in vari modi sono detti *operatori aritmetici*. Essi sono +, −, ★ e /. Il BASIC consente inoltre di manipolare stringhe in vari modi usando gli *operatori stringa*. Solo uno di essi abbina due stringhe ed è detto operatore di "concatenamento" ed è scritto come un segno +. L'operatore semplicemente aggiunge la seconda stringa al termine della prima in modo che

"PESCE" + "GATTO" = "PESCEGATTO"

Osservare la seguente sequenza di comandi e prevedere il risultato delle istruzioni PRINT. Quindi provare la sequenza sul computer; ricordarsi di lasciare uno spazio prima di ciascuna virgoletta di chiusura:

```
LET B$="DOG [SPACE] "
LET C$="BITES [SPACE] "
LET D$="MAN [SPACE] "
LET E$= B$+C$+D$
LET F$= D$+C$+B$
PRINT E$          [          ]
PRINT F$          [          ]
```

PRINT e LET sono i due comandi più frequentemente usati nel BASIC. Vale la pena di ricordare che quando si usa il **64** è possibile sostituire la parola PRINT con un simbolo: il punto di domanda (?). LET può essere omesso del tutto. Una sequenza valida è

```
A=5

B=17

?A,B
```

Il programma che viene fornito con questa unità è studiato per fare abbondante pratica con i comandi PRINT e LET ed è detto UNIT4DRILL.
È possibile interrompere il programma quando si è sicuri di aver perfettamente compreso l'uso delle variabili numeriche e stringa.

Esperimento 4.2 completato	

UNITA':5

È venuto il momento di esaminare i comandi memorizzati. Iniziamo mostrando che il **64** può realmente accantonare i comandi nella sua memoria e quindi richiamarli successivamente.
Accendere la macchina (oppure se è già funzionante con altro programma tipo SPEEDTYPE, fermarla battendo RUN STOP) e impartire il comando NEW (seguito come al solito dal tasto RETURN).

Questo comando ripulisce la memoria del **64** esattamente come un insegnate ripulisce la lavagna all'inizio della lezione. Non succederà nulla, salvo la comparsa della risposta READY. in quanto la memoria è tutta all'interno del computer.
Ora, battere il comando contrassegnato

10 PRINT 13+59

(NOTA: battere "uno zero" e non la O maiuscola)

e premere il tasto RETURN. Il solo risultato visibile è che la macchina sposta il cursore all'inizio della riga successiva. Il risultato della somma 13+59* non è calcolato nè visualizzato sullo schermo. Per contro, è successo qualcosa di invisibile: il **64** ha ricordato il comando e lo ha accantonato nella sua memoria interna.
Per verificare ciò, cancellare per prima cosa lo schermo (usando i tasti SHIFT e CLR HOME) e quindi impartire il comando

LIST

Se tutto è stato fatto nel modo corretto, sullo schermo dovrebbe ricomparire una copia del comando contrassegnato. Ciò dimostra che si trovava già nella macchina.
Finora, il nostro comando PRINT è stato memorizzato e richiamato ma non è stato ancora eseguito.

Non c'è nulla di speciale a proposito della somma 13+59. Qualsiasi altro comando PRINT avrebbe funzionato altrettanto bene in questo esempio.

Ma il computer ci deve ancora dire quanto fa 13+59. Per calcolare, battere

GOTO 10

ricordando di usare la lettera O maiuscola (e non la cifra zero) in GOTO.
Ciò dice al **64** di *eseguire* il comando contrassegnato "10". Esso procede in questo senso e finalmente compare la risposta. È possibile eseguire questa operazione quante volte si vuole. Un comando non viene distrutto quando viene listato o eseguito.
Il **64** ricorda molti comandi contemporaneamente. (Il limite è definito dalla dimensione della memoria: occorre un byte per contenere ciascun carattere di un comando più qualche bit di supervisione per il comando nel suo complesso). Ogni comando deve essere preceduto dalla propria etichetta o contrassegno e tutte le etichette devono essere diverse. La macchina memorizza e lista sempre i comandi in ordine crescente di etichette ed esegue questi comandi nell'ordine, a meno che non sia istruita a fare diversamente.
Provare battendo NEW

```
10 PRINT "PRIMA RIGA"
20 A=5
30 B=10
40 PRINT A;B;A+B
50 STOP
```

Ricordarsi di terminare ciascun comando con RETURN

Provare ora un LIST e quindi un GOTO 10 e controllare che i risultati siano quelli previsti. Lo STOP interrompe il **64** e visualizza un READY quando raggiunge la fine della sequenza di comandi.
La sequenza in cui i comandi sono memorizzati è quella esatta anche se li si batte in ordine diverso dai rispettivi numeri di label. Per esempio, se si fosse battuto

```
30 B=10
10 PRINT "PRIMA RIGA"
40 PRINT A;B;A+B
50 STOP
20 A=5
```

questi cinque comandi sarebbero sempre stati listati ed eseguiti nell'ordine 10, 20, 30, 40, 50.
Cancellare la memoria con un NEW e provare ancora.
Iniziare sempre con i numeri che vanno a incrementi di 10. Se si decide successivamente di inserire qualche comando tra quelli già scritti, l'operazione sarà possibile usando numeri di etichette intermedie come 15 o 38.
Perchè preoccuparsi di memorizzare i comandi? Ci sono due buoni motivi:

- I comandi che vengono richiamati dalla memoria interna del **64** sono eseguiti più velocemente di quelli che vengono battuti.

- I comandi che sono stati battuti una volta, possono essere eseguiti parecchie volte. Praticamente, ogni lavoro utile eseguito dal computer, comporta la ripetizione e vale quindi la pena di inserire i comandi nella memoria del computer dove vengono richiamati facilmente e velocemente.

Forse il modo più facile per ottenere la ripetizione è di memorizzare un comando contrassegnato GOTO. Si consideri il seguente programma:

```
10 PRINT "NORD"

20 PRINT "OVEST"

30 PRINT "SUD"

40 PRINT "EST"

50 GOTO 10
```

Quando viene avviato dall'etichetta 10, il **64** obbedisce ai primi quattro comandi in sequenza. Il successivo comando lo rimanda all'etichetta 10, cosicchè inizia la sequenza da capo. Esso continua così all'infinito e si ferma soltanto battendo RUN STOP o spegnendo la macchina.
Cancellare ora la memoria e battere il programma. Avviarla dando il comando iniziale

GOTO 10

Si vedrà la macchina obbedire alle righe del programma, molto più velocemente di quanto non sia possibile leggerle. È possibile rallentare la macchina premendo e tenendo abbassato CTRL (provare) oppure è possibile fermarla nel modo solito con il tasto RUN STOP.
A questo punto si vedrà effettivamente il vantaggio di usare i numeri di label distanziati di 10.
Si supponga ora di voler variare il programma in modo che comprenda le direzioni diagonali

NORD

NORD-OVEST

OVEST

SUD-OVEST

ecc.

Occorrono in questo caso nuove istruzioni da inserire tra quelle esistenti. Se le si numera 15, 25, 35 e 45 esse andranno nei posti giusti. Battere quanto segue:

```
15 PRINT "NORD-OVEST"

25 PRINT "SUD-OVEST"

35 PRINT "SUD-EST"

45 PRINT "NORD-EST"
```

Ora listare (LIST) il programma e controllare che le nuove righe siano state inserite tra le vecchie nei posti giusti. Eseguire il programma e vedere cosa succede.
Ora scrivere e provare un proprio programma sulle stesse righe. Se si usano caratteri grafici sulle stringhe invece delle lettere, è possibile creare sullo schermo qualche profilo interessante.
Il comando GOTO 10 che è usato per avviare il programma, ha un equivalente molto più comodo: RUN fa sì che il **64** inizi ad eseguire i comandi partendo da quello col numero minore.
Un ";" dopo un'istruzione di PRINT significa che, (durante l'esecuzione), alla fine il **64** non salterà all'inizio della riga successiva ma continuerà subito dopo il punto e virgola. (Naturalmente occorre sempre terminare ciascun comando con il tasto RETURN). Per contro, esso inizia una nuova riga sullo schermo soltanto quando raggiunge il margine di destra. Un semplice programma come quello che segue riempirà rapidamente l'intero schermo con disegni curiosi; provarlo e spiegarne l'azione.

```
10 PRINT "├─┤┬┴ ┌─┐ └─ ";

20 GOTO 10
```

Esperimento 5.1. completato	

Una sequenza di comandi che viene ripetuta più volte, è detta iterazione o loop. Un'interazione può includere diversi tipi di comandi compreso un LET che attribuisce un nuovo valore ad una variabile. Osservare questo programma e cercare di prevederne l'azione:

```
10 LET A = 1
20 PRINT A
30 LETA A=A+1
40 GOTO 20
```

Si immagini di essere cioè il computer e di eseguire esattamente ciò che il computer farebbe in questo caso, pazientemente, passo per passo. Scrivere cosa succede alla variabile A e ai suoi valori. Non procedere con la lettura fino a che non si è riflettuto e scritto la risposta.

(e così via)

Immettere ora il programma ed eseguirlo, tenendo abbassato il tasto CTRL per rallentarlo. (Ma non toccare CTRL fino a che non si è battuto RETURN dopo RUN).

Questo problema era abbastanza facile ma qui c'è in ogni caso la spiegazione di ciò che si è visto.

Il programma inizia eseguendo il comando contrassegnato 10, che dà alla variabile A il valore 1. Il successivo comando visualizza questo valore sullo schermo.

Il comando 30 sostituisce A con A+1. Questo equivale ad aggiungere 1 al vecchio valore di A, cosicchè il risultato (questa volta) è 2. Il successivo comando è un GOTO e fa sì che il **64** torni al comando 20. Il valore di A è visualizzato di nuovo ma stavolta è 2. La macchina continua a lavorare sull'intera sequenza, 20, 30, 40, ripetutamente, ma ogni volta il valore di A viene aumentato di 1. Ciò dà la sequenza 1, 2, 3...

Per fare un po' di pratica, cercare di prevedere le prime righe visualizzate da questi due programmi (ricordarsi che ★ significa moltiplicato per):

10 B=0	**10 A = 1**
20 PRINT B	**20 B = A★A**
30 B = B+3	**30 PRINT A,B**
40 GOTO 20	**40 A=A+1**
	50 GOTO 20

E ora controllare se si è previsto in maniera esatta.

È possibile fare lo stesso genere di cose con le stringhe. Provare questo programma:

```
10 X$ = "★"
20 PRINT X$
30 X$ = X$ + "="
40 GOTO 20
```

I successivi valori di X$ man mano che il programma procede nell'iterazione saranno ★, ★#, ★##, ★###, e così via. La stringa X$ diventa sempre più lunga e usa sempre più spazio sullo schermo ogni volta che viene stampata. Dopo circa 40 secondi, la stringa è così lunga che non entra più nella memoria della macchina (il numero massimo di caratteri ammesso è 255) e la macchina segnala una situazione anomala:

? STRING TOO LONG

ERROR IN 30

La riga ERROR in 30 significa che il comando che ha tentato di memorizzare la stringa errata era quello contrassegnato 30.
Ecco alcuni altri programmi sui quali fare previsioni.

10 A$ = "++"	10 A$ = "XY"
20 PRINT A$	20 PRINT A$
30 A$ ="A"+A$+"—"	30 A$ = A$+A$
40 GOTO 20	40 GOTO 20

Ricordarsi che se una lettera compare all'interno di una stringa, si tratta semplicemente di una lettera e non di un nome variabile. Così "X" non ha nulla a che fare con la variabile X$.
Come esercizio finale, scrivere un programma con una semplice iterazione e seguirlo esattamente per 1 minuto, controllando col cronometro. Quindi fermarlo, vedere dove è arrivato e *calcolare* quanti comandi la macchina ha eseguito in quel periodo di tempo. Ridurre la cifra in numero di comandi al secondo e scrivere qui la risposta.
Il quiz di auto-test per questa unità è detto UNIT5QUIZ.

Esperimento 5.2 completato

UNITA':6

Lo scopo di questo intero corso è di aiutare ad imparare a disegnare e costruire propri programmi. Per affiancare la crescente conoscenza di programmazione occorrerà una raccolta di tecniche o "strumenti" per organizzare il lavoro e aiutare a rimettere le cose in ordine se vanno male. Questa unità può essere considerata come una "scatola di attrezzi" o una "cassetta di pronto soccorso". Non vi si parlerà questa volta di programmazione come tale ma i suoi contenuti saranno utili in caso di emergenza. Leggere l'unità attentamente, cercare di conoscere le tecniche che essa descrive e riservare loro un posto permanente nella mente man mano che si procede col corso.
Se si dispone di un'unità a dischetti, è necessario inizializzare un nuovo dischetto prima di iniziare l'Esperimento 6.1.
Se si usa un registratore a cassetta per caricare il programma è possibile saltare questa sezione.
Come è noto, su un disco si possono memorizzare diversi programmi. Ogni disco ha un altro elemento: si tratta di un indice o "catalogo" che elenca i nomi e le dimensioni dei programmi memorizzati su quel disco.
Provare col seguente esperimento. Caricare il disco programma fornito insieme al corso e inizializzarlo nel modo solito. Quindi battere

LOAD"$",8 RETURN

LIST RETURN

Questi comandi richiamano nel **64** il catalogi o indice del disco e lo visualizzano sullo schermo, che si presenta più o meno come segue:

```
0 "ITB PROGRAMS          " DT 2A
12 "TEST PROG"              PRG
10 "HANGMAN"                PRG
13 "SPEEDTYPE"              PRG
```

e così via fino a

470 BLOCKS FREE

Vale la pena di studiare questo catalogo. La riga superiore, che compare in negativo, fornisce l'identità del disco stesso. In questo caso il nome — ITB PROGRAMS — è stato scelto dalla Commodore. "DT2A" è un numero di matricola pure pertinente al disco.
Successivamente c'è una riga per ogni programma. La prima entrata fornisce la dimensione del programma in blocchi, ciascuno dei quali contiene 128 byte cosicchè è possibile notare, ad esempio, che HANGMAN è lungo da 1280 byte. La seconda entrata dà il nome del programma e la terza — "PRG" — è identica per ogni programma.
La riga al termine del catalogo indica quanti sono i blocchi inutilizzati. ITB PROGRAMS ha ancora spazio per altri 47 programmi aventi la dimensione di HANGMAN o più, se sono più piccoli. La capacità di un disco completamente vuoto è di 664 blocchi.
Ora che si sa come sono strutturati i dischi programma, è possibile passare alla fase successiva.
Munirsi di un disco nuovo e vuoto del tipo adatto, preferibilmente acquistato dal rivenditore Commodore.
Assicurarsi che la tacca posta sulla sinistra quando si carica il disco sia libera e non coperta da un'etichetta argentata* come il disco programma ITB PROGRAMS.

* *L'etichetta rende impossibile all'unità a dischetti di registrare alcunchè sul disco e rappresenta un modo per impedire che i programmi del corso vengano accidentalmente distrutti. Dato che i propri programmi verranno registrati sul nuovo disco, non occorre alcuna protezione.*

Accendere l'unità a dischetti e caricare il disco. Battere

CLOSE 1 RETURN

OPEN 1,8,15, "N: nome disco,01" RETURN

dove "nome disco" è il titolo — qualsiasi — che si vuole attribuire al proprio disco, ad esempio PROGRAMMI DI GINO. In questo caso si dovrebbe battere

OPEN 1,8,15, "N: PROGRAMMI DI GINO,01"

Attendere circa un minuto, fino a che la spia rossa non cessa di lampeggiare quindi battere

LOAD"$",8 RETURN

LIST RETURN

Si ottiene un indice o catalogo vuoto.
Sullo schermo compare cioè la scritta

```
0"PROGRAMMI DI GINO"01 2A
664 BLOCKS FREE
```

È stato così formattato un disco. Ogni nuovo disco che viene acquistato deve essere formattato solo una volta nella sua vita ma in compenso occorre inizializzarlo ogni volta che viene caricato. Se si esegue la formattazione di un disco che è già stato usato per memorizzare programmi, si distrugge tutto ciò che vi è stato registrato. Fare quindi attenzione!
Estrarre il disco così preparato dall'unità, scriverne il titolo sull'etichetta e metterlo da parte.

ESPERIMENTO 6·1

Caricare ed eseguire il programma dell'Unità 6 denominato SENTENCES. Dare uno sguardo alle frasi "casuali" che esso visualizza. Queste dichiarazioni assurde sono costruite mediante una forma di "conseguenze" interne in cui ciascuna parola o frase viene scelta a caso da una breve lista di parole o frasi possibili. Qui non dobbiamo preoccuparci del modo in cui funziona il programma (quantunque in linea di principio sia abbastanza semplice), ma lo useremo come esempio per mostrare come listare, alterare e conservare grandi programmi.
Quando si sono viste abbastanza frasi, interrompere il programma con il tasto RUN STOP ed eseguire un LIST. Il programma è troppo ampio per potersi inserire interamente sullo schermo per cui, man mano che la lista viene eseguita, la maggior parte di essa scompare dalla parte superiore del video. Al termine, si possono vedere soltanto gli ultimi sei comandi.
Il linguaggio BASIC comprende alcune versioni speciali del comando LIST per tener conto di questa situazione. Ci sono cinque possibilità, che si dovrebbero provare man mano le si incontra:

- È possibile listare l'intero programma battendo LIST. Ciò, come è ora chiaro, ha taluni inconvenienti se il programma è troppo lungo.
- È possibile listare un comando scelto attribuendogli un proprio numero. Ad esempio

  LIST 1100

  visualizza il comando 1100 (e nessun altro).
- È possibile listare tutti i comandi fino a un dato numero di identificazione inserendo un segno — davanti al numero. Così

  LIST —80

  mostra tutti i comandi dall'inizio del programma fino a quello contrassegnato 80.
- È possibile chiedere tutti i comandi da un dato numero fino alla fine del programma inserendo un segno — dopo il numero:

  LIST 9090—
- Infine è possibile listare tutti i comandi tra due numeri qualsiasi citando entrambi i numeri di identificazione:

  LIST 2000 — 2090

Ora usare alcuni di questi tipi di comandi LIST per osservare le varie parti del programma. Si noterà rapidamente che i numeri di label o di identificazione non sempre vanno a incrementi di 10 e ciò in quanto il programma è stato modificato molte volte dopo che è stato scritto.
All'inizio del programma e in parecchi altri punti si vedranno comandi con la parola chiave REM seguita da descrizioni in inglese. REM è un'abbreviazione per "commento". Le righe che contengono commenti non svolgono alcun ruolo nel programma, ma sono incluse soltanto per rendere la lettura del programma stesso più facile. Quando si inizia a scrivere programmi complicati occorre sempre usare abbondanti REM per spiegare ciò che si sta facendo.
Una volta comprese a fondo le varie forme del comando LIST, provare a fare qualche esperimento per vedere cosa succede se:

(a) Il comando a cui si fa riferimento non è presente.

(provare LIST 650)

(b) I numeri di label sono nell'ordine sbagliato

(provare LIST 1100 — 1000)

(c) Il comando LIST è incluso in un programma. Battere NEW, quindi battere questo programma ed eseguirlo.

```
10 PRINT "PROVA LIST"
20 LIST
30 GOTO 10
```

Esperimento 6.1 completato	

ESPERIMENTO 6·2

Questo esperimento spiega come i programmi possono essere modificati e alterati. Per cominciare, si eseguiranno modifiche ad un programma originariamente scritto da qualcun altro, ma successivamente la maggior parte delle modifiche riguarderanno programmi propri.
Ci sono tre tipi di modifiche che è possibile apportare ad un programma:

(a) Rimozione di comandi esistenti

(b) Aggiunta di nuovi comandi

(c) Correzione o sostituzione di comandi esistenti

Rimozione di comandi esistenti

Ci sono cinque modi per liberarsi di un comando contrassegnato o identificato nella memoria del **64**, ma tre di essi comportano la cancellazione o la modifica dell'intero programma e sono pertanto di effetto abbastanza drastico.

Un intero programma può essere cancellato

- Spegnendo il **64**

o • Battendo NEW
o • Caricando un nuovo programma dalla cassetta a nastro o dal disco

Un singolo comando può essere rimosso

- Battendone il solo numero della label o identificazione
- Battendo un altro comando con lo stesso numero di label o identificazione

Ricaricare il programma SENTENCES e cancellarne alcune righe che contengono la parola chiave REM. Controllare che la cancellazione abbia funzionato listando (LIST) un'appropriata parte del programma sia prima che dopo.

Aggiunta di nuovi comandi

Un nuovo comando può essere aggiunto ad un programma esistente battendolo con un adatto numero di label. Il comando è inserito nel punto determinato dal numero della label o identificazione.
Abbiamo già fatto pratica nell'inserire i comandi nell'Unità 5 ma occorrerà sfruttare questa opportunità per inserire alcuni comandi REM. Assicurarsi di non sostituire un'istruzione esistente altrimenti il programma non funzionerà.

Alterazione di comandi esistenti

Il modo più drastico per alterare un comando consiste nel ribatterlo, usando lo stesso numero di etichetta o di identificazione.
Si provi con un esempio di questo genere. Iniziare sostituendo la riga del copyright (label 5) con una contenente un altro nome. Un esempio potrebbe essere il seguente:

Cercare di cambiare altre righe ma limitarsi a quelle con le parole chiavi REM altrimenti si corre il rischio di danneggiare il programma impedendogli di funzionare correttamente.
Il programma è come una cellula vivente: le mutazioni casuali sono quasi sempre critiche e solitamente fatali.
Quando una riga ha bisogno di una piccola modifica, è spesso più facile variare l'originale (che è già sullo schermo) che non batterne una nuova versione. Ciò avviene con il cursore ed eventualmente con il tasto INST DEL. Quando le modifiche sono completate, il tasto RETURN fa sì che il **64** registri il nuovo comando in luogo del vecchio.
Si supponga di voler alterare la riga 100 in modo che appaia come:

100 REM MAIN STUPID SENTENCE GENERATOR

Listare (LIST) il comando 100 e inserire il cursore sulla S di SENTENCE e inserire 7 spazi vuoti (usare i tasti SHIFT e INST DEL). Quindi battere la parola STUPID, controllare che tutto sia corretto e battere RETURN. Eseguire un altro LIST 100 come controllo.
Provare altre alterazioni di questo tipo, sempre limitandosi ai comandi REM. Ricordarsi che se non si batte RETURN dopo aver cambiato una riga con il cursore, la macchina non registra alcuna variazione.
Battere ora RUN al termine del programma. Se non funziona, si è certamente compiuto un

errore nella correzione, ad esempio cancellando o alterando inavvertitamente un'istruzione. Non preoccuparsi — è una cosa abbastanza comune. Basta ricaricare il programma dal nastro sul disco.
Si sarà osservato che il programma SENTENCES fa delle dichiarazioni su personaggi ben noti. La lista delle possibili scelte è molto breve: esse sono nei comandi 9070 (per gli uomini) e 9100 (per le donne). Per la parte finale di questo esperimento, si procederà ad alterare queste liste in modo che il programma crei delle frasi sulla famiglia e sugli amici.
Ciascuno dei due comandi 9070 e 9100 ha la parola chiave DATA. Questa è seguita dalla lista di nomi, separati da virgole. L'ultimo nome è seguito dalla virgola e dalla lettera Z.*
Ci possono essere nomi a piacere. Se i nomi occupano più di quattro righe, usare un secondo comando DATA (con un numero di etichetta maggiore di 1 rispetto al primo). Possono anche essere usati un terzo e un quarto comando DATA ma soltanto l'ultimo comando DATA in ciascun gruppo ha bisogno della Z al termine.
Alcuni possibil alternative per le righe 9070 e 9100, potrebbero essere:

```
9070 DATABILL,GEOFFREY,PERCEVAL,MR.SOPHOC
LES,THE HEADMASTER,Z
9100 DATAGRANNY,SUSAN,VIOLET,MRS.PINKERTO
N,THE GYM MISTRESS, AUNTIE FLO, RACHEL
9101 DATAPENNY,KATE,LAURA, FRANCES, NORAH, V
ICKY,Z
```

Una volta effettuate queste modifiche, eseguire il programma di nuovo. Se esso risulta un completo non senso, controllare se è stata posta una virgola tra ciascun nome (ma non due virgole) e che l'ultimo nome sia seguito da una Z.
Una volta presa l'abitudine ad effettuare modifiche, è possibile applicare la propria fantasia alle altre liste di parole nel programma. Esse sono:

9000 Azioni che la gente fa a se stessa (verbi intransitivi)

9010 Azioni che la gente fa con altri (verbi transitivi)

9020 Azioni che la gente fa con gli abiti (verbi transitivi)

9030 cose indossate dagli uomini

9040 cose indossate dalle donne

L'uso di Z al termine del comando DATA è una caratteristica soltanto di questo programma, non del BASIC in generale. I comandi DATA nella maggior parte degli altri programmi, non richiedono la Z al termine.

9050 Una lista di avverbi e frasi avverbiali che descrivono azioni che la gente fa reciprocamente

9060 Una lista di avverbi e frasi avverbiali che descrivono azioni che la gente fa a se stessa

9070 Nomi di uomini

9080 Aggettivi che descrivono uomini

9090 Vari tipi di uomini

9100 Nomi di donne

9110 Aggettivi che descrivono le donne

9120 Vari tipi di donne

Alterare le liste in qualsiasi modo a piacere, facendo però in modo di mantenerle coerenti. Se si altera 9020 nelle azioni che riguardano i cibi, occorre anche alterare 9030 e 9040 di conseguenza, altrimenti si potrebbero avere frasi di questo genere.

SUSAN ATE HER WELLINGTON BOOTS
(Susan ha mangiato i suoi stivali Wellington)

Esperimento 6.2 completato	

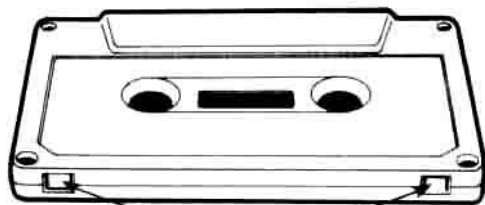
ESPERIMENTO 6·3

Una volta alterato il programma SENTENCES per esprimere concetti divertenti su amici, è possibile conservare la nuova versione per mostrarla in occasione di feste, ecc. Questa sezione spiega come conservare il programma sulla cassetta a nastro.
Procurarsi un nastro vuoto o uno contenente qualcosa che non si ha interesse a conservare. Deve trattarsi di un nastro di buona qualità ed essere il più breve possibile: si userà infatti soltanto circa 1 minuto della durata del nastro e non vale la pena di spendere di più per una durata maggiore. Le cassette della Commodore sono l'ideale.
Caricare il nuovo nastro nell'unità a cassetta in luogo della cassetta SENTENCES e riavvolgerlo.
Sbloccare tutti i tasti sull'unità a cassetta, quindi interrompere il programma SENTENCES e battere

SAVE "FAMIGLIA" RETURN

(è possibile usare qualsiasi nome invece di FAMIGLIA).
La macchina risponde

PRESS RECORD AND PLAY ON TAPE
(Premere RECORD e PLAY)

Seguire queste istruzioni premendo entrambi i tasti sul registratore contemporaneamente.
Se il tasto RECORD non si abbassa, controllare la cassetta per assicurarsi che non siano state tolte le linguette di protezione scrittura. Queste linguette si trovano nella parte posteriore della cassetta come indicato in figura:

LINGUETTE

Se le linguette sono asportate, è impossibile registrare sul nastro. Lo scopo è di proteggere le registrazioni importanti che non devono essere distrutte per cui in questo caso occorre procurarsi un altro nastro.

Se tutto procede regolarmente, la macchina dice

SAVING FAMIGLIA

e in un momento successivo

READY.

In teoria il programma è ora registrato, ma vale la pena di controllare. Varie cose possono essere andate storte: è possibile essersi dimenticati di riavvolgere il nastro o di premere il pulsante RECORD, oppure il nastro stesso potrebbe avere un piccolo difetto che gli impedisce di eseguire una copia corretta del programma. Queste cose non dovrebbero succedere, ma in pratica avvengono!
Per controllare il nastro, riavvolgerlo, quindi battere

VERIFY "FAMIGLIA" RETURN

La macchina risponde

PRESS PLAY ON TAPE

Premere il pulsante PLAY (non il pulsante RECORD stavolta). La macchina cerca quindi il programma su nastro e lo controlla a fronte di ciò che c'è nella memoria. Naturalmente non si devono fare modifiche tra i comandi SAVE e VERIFY.
Se tutto va bene, i messaggi che si vedranno comparire sono:

```
VERIFY "FAMIGLIA"
PRESS PLAY ON TAPE
OK
SEARCHING FOR FAMIGLIA
FOUND FAMIGLIA
VERIFYING
OK
```

Se la macchina trova un errore oppure non arriva a FOUND FAMIGLIA, ricominciare daccapo e provare il comando SAVE. Se il difetto persiste, provare un altro nastro (o provare l'altro lato del primo). Se non è possibile ancora far funzionare il sistema, portare il **64** e la cassetta al rivenditore per un controllo.
Una volta che il programma è stato salvato (SAVE), può essere riposto altrove e caricato (LOAD) in qualsiasi momento, con un comando tipo

LOAD "FAMIGLIA"

Un programma non deve essere perfetto per poter essere salvato (SAVE).

Per salvare il programma su un disco, rimuovere il disco programma ITB PROGRAMS e caricare quello formattato all'inizio dell'unità. Inizializzarlo battendo

CLOSE 1 RETURN

OPEN 1,8,15,"I" RETURN

Registrare ora la propria versione del programma con il comando

SAVE"nome del programma",8 RETURN
dove "nome programma" è il titolo che è stato scelto per il programma. Ad esempio

SAVE"FAMIGLIA",8 RETURN
Se tutto procede regolarmente, la macchina risponde con
SAVING FAMIGLIA
e un attimo dopo con
READY.
Per assicurarsi che il programma sia stato registrato correttamente, battere

VERIFY"nome programma",8 RETURN (dove "nome programma" è il nome attribuito a propria scelta). La macchina risponde con
SEARCHING FOR nome programma
VERIFYING
OK
READY.
Questo processo funziona sempre praticamente come è descritto, in caso contrario, riprovare da capo attentamente, fino alla fine. Se non riesce ancora, chiedere consiglio al rivenditore.
Una volta salvato (SAVE) il programma ed averlo verificato (VERIFY), caricarlo e listare il catalogo, in cui dovrebbe ora comparire il programma testè caricato.
Una volta che un programma è stato salvato (SAVE) può essere ricaricato nel modo solito da un comando del tipo

LOAD"FAMIGLIA",8 RETURN
Disponendo di un disco formattato, è possibile continuare ad inserirvi programmi fino al termine dello spazio disponibile. Di tanto in tanto può essere necessario liberarsi di un programma e sostituirlo con una nuova versione avente lo stesso nome. Se si impartisce il comando
SAVE"nome programma",8
e sul disco c'è già un programma con quel nome, la spia rossa inizia a lampeggiare ma non succede nulla. Per liberarsi del vecchio programma occorre usare due caratteri in più davanti al nome del programma @ : (due punti).
Il comando di salvataggio si presenta ora come segue:
SAVE"@: nome programma",8
È opportuno far seguire questo tipo di comando SAVE con il comando speciale
OPEN 1,8,15,"V"
ed attendere che la spia rossa si spenga. Ciò dà alla macchina la possibilità di riunire lo spazio liberato dalla rimozione del programma in modo da poterlo riutilizzare.
Un programma non deve essere perfetto per poter essere salvato (SAVE).
Se si scrive un programma molto lungo (o addirittura se se ne copia uno da un libro), vale la pena di eseguire un comando SAVE ogni mezz'ora o giu di lì. Ciò in quanto la memoria del **64** non è altrettanto affidabile del nastro in un cassetto. È improbabile che la macchina si rompa, ma possono verificarsi altri incidenti. È successo che il fulmine abbia danneggiato le informazioni nella memoria di un computer, potrebbe esserci una mancanza di corrente oppure, qualcuno potrebbe camminare sul cavo di alimentazione e staccarlo dalla presa. Se si perdono sei ore di lavoro è causa di un tale incidente, è possibile sentirsi piuttosto frustrati. Se, per contro, sono state effettuate regolari copie della memoria ogni mezz'ora, è possibile ricaricare la versione più recente e procedere soltanto con una piccola perdita di tempo.
Per rendere il sistema assolutamente sicuro, occorre salvare (SAVE) su due diversi nastri o dischi alternativamente. In tal caso anche se la macchina si ferma durante un SAVE, con la metà della versione coperta da metà della nuova, si è sempre protetti.

Esperimento 6.3 completato	

ESPERIMENTO 6·4

In questa sezione si parlerà di una trappola sottile e pericolosa che è in attesa dei programmatori 64 e si dirà come uscirne se vi si cade.
Per cominciare, si cercherà di mettere per iscritto un semplice esempio della trappola. Battere NEW per cancellare la memoria e quindi immettere il seguente programma inserendo accuratamente tutti gli spazi indicati e osservando lo schermo mentre si batte.

```
10 PRINT"A FRIGHTFUL AND APPALLING TRAP"
20 GOTO 10
```

Ora, eseguire LIST e controllare il programma, che dovrebbe comparire esattamente come indicato.
Il programma, battendo RUN, visualizzerà il messaggio

A FRIGHTFUL AND APPALLING TRAP

ripetutamente fino a che non viene fermato. Provare e osservare. È probabile invece che compaia

A FRIGHTFUL AND APPALLING TRAP 20

? SYNTAX ERROR IN 10

READY.

Anche se il programma funziona correttamente occorre leggerlo per scoprire come si è fatto per evitare la trappola.
Il motivo per cui la macchina non è stata in grado di eseguire il programma (supponendo che ciò sia quello che ha fatto), non è per nulla ovvio. Si potrebbe mostrare il programma ai più grandi esperti di BASIC del mondo che non vi vedrebbero alcunchè di sbagliato.
La difficoltà sorge a causa della larghezza dello schermo del 64. All'interno della macchina, qualsiasi comando BASIC può avere una lunghezza massima di 88 caratteri. Lo schermo è largo soltanto 40 caratteri cosicchè la versione *visualizzata* di un comando può distribuirsi su un massimo di 3 righe di schermo.
Quando si batte un comando e il cursore raggiunge la fine di una riga dello schermo, il sistema lo sposta all'inizio della riga successiva, ma presume ancora che stia battendo lo stesso comando. Un comando viene terminato soltanto dal tasto RETURN.
Se si cade nella trappola (come si è supposto che sia accaduto), ecco cosa succede:
Si batte il primo comando (che è stato opportunamente studiato per riempire l'intera riga dello schermo). A questo punto il cursore si trova all'inizio della riga successiva e naturalmente si batte il comando successivo, terminandolo con un RETURN. Dato che non si è terminata la prima riga con un RETURN, il sistema pensa che entrambe le righe facciano parte dello stesso comando e cioè

10 PRINT "A FRIGHTFUL AND APPALLING TRAP" 20 GOTO 10

Questo "comando" non è scritto in BASIC corretto e dà luogo a un errore di sintassi quando la macchina cerca di eseguirlo.
Questo tipo di errore è particolarmente difficile da trovare, a meno che non si sappia che cosa si sta cercando. È molto improbabile notare l'errore quando si batte il programma — anche i programmatori esperti spesso dimenticano di terminare i comandi con RETURN se il cursore si trova all'inizio di una nuova riga. Se si lista (LIST) il programma o anche solo la sezione che comprende l'errore, il comando difettoso assomiglia esattamente a due comandi corretti e il difetto è invisibile.
Fortunatamente l'errore può essere individuato listando (LIST) il comando in cui è segnalato l'errore. Se si batte LIST 10, escono le righe 10 e — apparentemente — 20! Ciò deve essere sbagliato dato che si è chiesta soltanto la riga 10. Per correggere l'errore, ribattere entrambi i comandi completamente, ricordando di terminare ciascuno di essi con RETURN.

Riassumendo:

(a) Terminare sempre qualsiasi comando con RETURN, ovunque si trovi il cursore.

(b) Se il 64 segnala un errore in un comando e non è possibile vedere alcunchè di sbagliato, listarlo (LIST) e controllare se funziona sul successivo comando nel programma.

Esperimento 6.4 completato	

UNITA':7

I programmi scritti nell'Unità 5 erano indisciplinati. Una volta avviati, occorreva un'azione drastica per interromperli e, se lasciati fare, avrebbero continuato indefinitamente. Questa unità tratta del modo in cui controllare i programmi e interromperli dopo un certo periodo di tempo.
Gli argomenti descritti in questa unità sono fondamentali per la programmazione. Una volta padroneggiati completamente, si sarà compiuto il passo più grande per diventare programmatori. Leggere l'unità lentamente e attentamente e se ci sono dei dubbi su qualche punto, tornare indietro e rileggere di nuovo. Vale la pena di far ciò in quanto, una volta compresi questi argomenti, si potrà fare molta strada.
Il controllo dei programmi dipende da un concetto chiave che per qualcuno potrebbe essere nuovo: *la condizione.*
Quando la gente parla, espone delle affermazioni che sono vere e che almeno si suppone vengano prese per vere:

"Il mio treno si è rotto per strada"

"Ti amo".

Una condizione è un tipo speciale di dichiarazione che non è necessariamente vera ma che potrebbe ugualmente anche essere falsa. In inglese, si usano le condizioni dopo la parola "if". Nelle seguenti frasi, le condizioni sono evidenziate:

"Se (If) **l'ultimo treno è partito,** occorrerà trascorrere la notte a Milano".

"Se (If) **il programma non funziona,** occorre trovare l'errore e ripararlo".

Chi pronuncia queste frasi non sta insistendo sul fatto che il treno sia partito o che il programma realmente non funzioni; egli semplicemente non lo sa e sta preparando dei piani di conseguenza.
Una condizione può risultare vera o falsa senza che per questo chi la esprime venga chiamato bugiardo.
Anche nel BASIC, le condizioni vengono dopo la parola chiave IF. Esse coinvolgono i vari "oggetti" usati nei programmi: variabili numeriche, variabili stringa, numeri e stringhe. Le condizioni, che in ogni caso possono essere vere o false, sono costruite utilizzando una delle sei relazioni.
Ciò è meglio illustrato dall'esempio:

Si consideri la condizione BASIC:

A < 5

(dove < è un segno che significa "minore di").
Questa condizione è vera se il valore della variabile A è realmente minore di 5 (ad esempio 0 o 3 o 4.98). È *falsa* se A vale 5 o più di 5.
Un altro esempio stavolta con l'uso delle stringhe:

N$ <> "JIM"

(dove <> significa "è diverso da").
Questa condizione è vera se la variabile N$ ha qualsiasi valore salvo "JIM"; perciò è vera se N$="JACK" o N$="JIMMY". È falsa solo se N$ è effettivamente "JIM".
La serie completa di relazioni che è possibile usare nel BASIC comprende:

= (uguale a)

< (minore di)

> (maggiore di)

<> (diverso da)

<= (non maggiore di)

>= (non minore di)

Le relazione <>, <= e >= sono ciascuna battute con due operazioni sui tasti. Questi simboli possono essere più familiari nelle forme ≠, ≤, e ≥ ma i progettisti del BASIC hanno dovuto accettare il fatto che le tastiere dei computer non portano solitamente questi segni.
Le relazioni possono essere tutte usate sia tra coppie di numeri che tra coppie di stringhe per porre delle *condizioni.* I numeri e le stringhe possono essere rappresentati da variabili appropriate.

5>4 è *vero* in quanto 5 è maggiore di 4

7<=6 è *falso* in quanto 7 è maggiore di 6

se A = 10 e B =7,

A <=B è *vero* e così B<=7.

Quando vengono usate relazioni tra le stringhe, queste implicano l'ordine alfabetico (come da dizionario) in modo che

"CANE" > "GATTO" è vero,

e "GIOVANNI" > "GIOVANNINO" è falso.

ESPERIMENTO 7·1

Si supponga che il computer abbia eseguito le seguenti tre istruzioni:

LET A$ = "JOAN"

LET X = 5

LET Y = 7

Elaborare la tabella seguente e contrassegnare ciascuna condizione come falsa o vera:

Condizione	Valore (vero o falso)
X < 7	
X >= 5	
A$<>"X"	
Y <> X	
A$ < "FRANCES"	
A$ > "JOAN"	
Y = 8	

Le quantità su entrambi i lati della relazione possono essere espressioni, esattamente come nei comandi LET. Le espressioni possono essere complesse a piacere, ma la cosa importante è il confronto fra uguali: una condizione che ha un numero su un lato ed una stringa sull'altro fa si che il **64** si fermi e segnali un errore.
Si supponga che i valori di A$, X e Y siano gli stessi dell'esempio precedente ed elaborare ciascuna delle seguenti condizioni:

Condizione	Valore (vero o falso)
A$+"NE"<>"JOANNE"	
5 > X	
X + Y <> 13	
X + 2 = Y	

Controllare ora le risposte, che sono indicate nell'Appendice B.

Esperimento 7.1 completato	

ESPERIMENTO 7·2

Lo strumento principale di controllo in BASIC è il comando IF. Esso si compone di una parola chiave IF, una condizione, la parola THEN e un numero di label. È molto simile a GOTO, ma con una differenza: il salto avviene soltanto se la condizione è *vera*.

Un esempio del comando IF è

```
IF X$ <> "ABBBB" THEN 20
```

Qui la condizione è X$ <> "ABBBB" e l'intero comando dice alla macchina di saltare a 20 se (if) X$ è diverso da "ABBBB". Se questa condizione è falsa, la macchina continua ad eseguire i comandi nel loro ordine numerico.
La maggior parte delle persone reagisce a questo comando pensando che sia leggermente assurdo. "X$ è diverso da quella stringa con le A e le B" si pensa, "o non lo è. Tutto dipende da che cosa c'è prima ma in ogni caso quando il programmatore ha scritto quel comando IF, doveva saperlo"!
Questo punto di vista è comprensibile e plausibile ma è sbagliato. Potrebbero esserci due motivi completamente diversi:

- Si supponga che il comando IF sia in un'iterazione in cui una qualche variabile cambia il valore ad ogni passaggio. La condizione potrebbe anche essere vera per alcuni dei valori ma non per altri.

- Si supponga di nuovo che si stia scrivendo un programma che qualcun altro dovrà usare. In tal caso non si sa che cosa farà l'utente di tale programma, ma le azioni del programma dovranno in ogni caso dipendere da ciò che l'utente farà effettivamente. Ad esempio: occorre fare in modo che i diversi programmi del quiz rispondano in modo coerente alle risposte quantunque non si abbia idea di cosa si risponderà alle varie domande.

L'inserimento di un'istruzione IF in un'iterazione fornisce un modo interessante per interromperla dopo averla ripetuta un certo numero di volte. Battere ed eseguire quanto segue:

```
10 X$ = "A"
20 PRINT X$
30 X$=X$ + "B"
40 GOTO 20
50 STOP
```

Questo programma viene eseguito riempendo lo schermo con stringhe di B sempre più lunghe fino a che non si esaurisce tutto lo spazio. Lo STOP alla riga 50 non viene mai raggiunto.
Ora, fermare il programma e sostituire la riga 40 con

```
40 IF X$ <> "ABBBB" THEN 20
```

Quando si esegue il programma, sullo schermo compare

```
A
AB
ABB
ABBB
```

e poi il programma si ferma!

Il motivo sta nella condizione X$ <> "ABBBB". Quando il programma ripete l'iterazione, la condizione è all'inizio vera (in quanto X$ è AB e quindi ABB, che sono tutti diversi da ABBBB). In ciascuno di questi casi il comando IF si comporta come un semplice GOTO 20 e istruisce la macchina a compiere un'altra iterazione. Alla fine, X$, arriva a ABBBB. Ma la condizione è ora falsa: il salto non si verifica e la macchina arriva alla fine del programma dove si ferma.
Ora, provare a cambiare la condizione in vari modi ed osservare l'effetto durante l'esecuzione del programma. Qualsivoglia stringa si usi, assicurarsi che la condizione risulti al limite falsa altrimenti il programma non si fermerà mai.
Le possibili condizioni da provare sono:

```
X$<>"AB"
X$<>"ABBBBBBBBBBB"
X$<"ABBA"
```

La stessa tecnica di controllo può essere usata con le variabili numeriche.

Battere

```
10 P = 0
20 PRINT P, P*P
30 P=P + 1
40 GOTO 20
50 STOP
```

Eseguire questo programma e vedere cosa fa, fermarlo e cambiare la riga 40 in modo che appaia come segue:

```
40 IF P < 11 THEN 20
```

Eseguire ora di nuovo il programma. Esso visualizza due colonne di cifre che sembrano familiari e che potrebbero essere utili a chi non conosce a memoria i quadrati dei numeri. Trattandosi di un programma funzionante c'è soltanto una cosa sbagliata: la visualizzazione non è contrassegnata (mancano cioè le label) e il suo significato non è immediatamente ovvio salvo a chi ha scritto il programma. È possibile rimediare aggiungendo un titolo o una riga all'inizio che identifica ciascuna colonna come segue:

NUMERO	QUADRATO
0	0
1	1
2	4
3	9
. .	. . . . .

ecc.

Chiaramente il titolo deve essere visualizzato prima di qualsiasi numero o del relativo quadrato per cui il comando che lo visualizza deve venire per primo. Dato che è già stata usata la label 10 e che sarebbe troppo laborioso cambiare tutte le label dell'intero programma, la decisione giusta consiste nell'usare la label 5. Il comando è di per sè un PRINT con due stringhe: "NUMERO" e "QUADRATO". Le virgole tra le stringhe assicurano che la spaziatura corrisponda a quella esistente tra le colonne di cifre. L'intero programma risulta ora come segue:

```
 5 PRINT "NUMERO", "QUADRATO"
10 P=0
20 PRINT P,P★P
30 P=P+1
40 IF P<11 THEN 20
50 STOP
```

Eseguire il programma in questa forma ed esaminare il risultato.
Se si desidera una riga vuota tra il titolo e la prima fila di cifre? Il comando PRINT (non seguito da qualsiasi valore o stringa), darà una riga vuota cosicchè provare aggiungendo il comando

```
7 PRINT
```

Fra pochi minuti verrà chiesto di scrivere alcuni programmi. Prima di iniziare, occorre però dare uno sguardo attendo ai programmi già eseguiti e ricavare alcune conclusioni generali. I programmi esemplificativi sono:

(1)

```
10 X$="A"
20 PRINT X$
30 X$=X$+"B"
40 IF X$ <>"ABBBB" THEN 20
50 STOP
```

(2)

```
 5 PRINT "NUM","SQUARE"
 7 PRINT
10 P=0
20 PRINT P,P★P
30 P=P+1
40 IF P < 11 THEN 20
50 STOP
```

Se si trascurano i comandi del titolo (5 e 7) nel secondo, entrambi i programmi sembrano seguire un certo profilo. In ciascun caso:

1. C'è una variabile che cambia regolarmente man mano che l'iterazione viene ripetuta. Si vedrà che è X$ nel primo programma e P nel secondo. In generale, questa è chiamata una variabile di *controllo* e può essere una stringa o un numero.

2. C'è un comando che dà alla variabile di controllo il suo *valore iniziale*. Questo comando è al di fuori dell'iterazione (cioè non viene ripetuto ma viene eseguito soltanto una volta).

3. c'è un comando che viene eseguito per ogni valore nella variabile di controllo. Negli esempi, questi sono i comandi PRINT.

   PRINT X$

   e PRINT P,P★P

   In pratica, questa parte dell'iterazione può essere ampliata per includere qualsiasi numero di comandi, tutti eseguiti per ciascun valore della variabile di controllo. Questo gruppo è detto *corpo* dell'iterazione.

4. C'è un *incremento* o quantità di cui la variabile di controllo cresce ogni volta che viene ripetuta l'iterazione. Nei nostri esempi, X$ cresce aggiungendo una "B" e P viene aumentato di 1. Sono possibili altri incrementi; per esempio una stringa potrebbe crescere di 5 simboli alla volta o un numero potrebbe aumentare a incrementi di 2 (o di qualsiasi altro valore). Potrebbe anche iniziare con un valore alto e scendere o "decrementare" contando alla rovescia. L'iterazione comprende sempre un comando che sposta la variabile di controllo di un passo ogni qualvolta viene eseguita.

5. C'è un valore finale per la variabile di controllo. Quando l'iterazione viene eseguita con questo valore, la ripetizione deve cessare. L'ultimo comando nell'iterazione è un comando IF, con una condizione che è vera se l'iterazione deve essere ancora eseguita, ma falsa quando la variabile di controllo ha superato il suo valore finale.

Nella tabella che segue, si esamini ciascun programma e s'inserisca il nome della variabile di controllo, il valore iniziale, il valore finale, l'incremento e il numero delle iterazioni eseguite. Per far ciò, spesso è di aiuto indicare il valore della variabile controllata nella prima, seconda, terza... iterazione e vedere quanti valori ci sono fino al raggiungimento del valore finale.

	Variabile di controllo	Valore iniziale	Valore finale	Incremento	N. dei passaggi del ciclo
10 X$="A" 20 PRINT X$ 30 X$=X$+"B" 40 IF X$<>"ABBBB"THEN 20 50 STOP	X$	"A"	"ABBB"	"B"	4
10 P=0 20 PRINT P,P*P 30 P=P+P 40 IF P < 11 THEN 20 50 STOP	P	0	10	+1	11
10 Y$="Z" 20 PRINT Y$ 30 Y$=Y$+"XY" 40 IF Y$ <>"ZXYXYXY"THEN 20 50 STOP					
10 R=5 20 PRINT R, R/8 30 R=R+3 40 IF R < 17 THEN 20 50 STOP					
10 C=27 20 H=30-C 30 PRINT C,H 40 C=C-5 50 IF C > 2 THEN 20 60 STOP					

Una volta completata la tabella, controllare le risposte a fronte di quelle indicate al termine del manuale (Appendice B).

Esperimento 7.2 completato	

Quando si costruisce un programma, occorre iniziare creandone una qualche traccia e quindi scrivendo l'intero programma su un pezzo di carta. Usare gomma e matita! Alcuni compongono i programmi direttamente sulla tastiera del computer, ma questo metodo è adatto soltanto per i geni e per gli idioti — è chiaramente non consigliato per le persone normali. La ragione è abbastanza semplice: se s'inizia senza avere le idee chiare su ciò che si deve fare, si hanno altrettante probabilità di successo di un costruttore che imposta una casa senza disegni, creando l'architettura man mano che procede. Quel costruttore potrebbe anche produrre un gioiello architettonico, ma più probabilmente terminerà con una stamberga che cadrà a pezzi al primo temporale.

Quando si progetta un'iterazione per un programma, occorre decidere tutti gli elementi essenziali.

Questi comprendono il nome e il tipo della variabile controllata, i valori iniziali e finali, l'incremento e i dettagli per il corpo dell'iterazione. Una volta chiariti questi punti, è possibile metterli insieme nel profilo standard.

Ecco un esempio elaborato.

Una sterlina vale all'incirca 2350 lire italiane. Si tratta di costruire una tabella che dia l'equivalente italiano degli importi da 5 a 75 sterline, procedendo a incrementi di 5 sterline.

E cioè:

Lire sterline	Lire italiane
5	11750
10	23500
....	....

e così via

Pensiamo per prima cosa all'iterazione. La variabile di controllo chiaramente sarà un numero che si potrebbe chiamare PS (che sta per Pounds Sterling - Lire sterline). Il valore iniziale sarà 5 e il valore finale 75 e l'incremento 5. Il corpo dell'interazione deve stampare un valore in Lire sterline e il corrispondente valore in lire che è 2350 volte tanto.

Si possono quindi scrivere gli elementi della nostra iterazione, che sono:

PS=5 (Definisce il valore iniziale)

PRINT PS,2350★PS (Corpo)

PS=PS+5 (Incrementa PS)

IF PS<80 THEN (Controlla se il valore finale è superato)

STOP (Interrompe il programma)

Il numero di label che segue THEN, viene lasciato in bianco in quanto non si sa cosa succederà.
Prima di scrivere l'intero programma, occorre considerare il titolo. Adatti comandi potrebbero essere:

```
PRINT"£","LIRE"
```

e PRINT (Per ottenere una riga vuota)

È ora possibile assemblare le varie parti e scrivere l'intero programma:

```
10 PRINT"£","LIRE"
20 PRINT
30 PS=5
40 PRINT PS,2350★PS
50 PS=PS+5
60 IF PS <80 THEN 40
70 STOP
```

A rischio di risultare noiosi, è necessario ripetere: non cercare di abbreviare o di evitare il processo iniziale di disegno: non improvvisare il programma direttamente sul computer. Così facendo non si diventerà mai programmatori.
Provare ora questi esempi:

1. Scrivere un programma che visualizza un profilo di asterischi, cioè:

   ★
   ★★
   ★★★
   ★★★★
   ....
   fino a
   ★★★★★★★★★★

2. Scrivere un programma che dia l'equivalente di importi in dollari americani (dollars) per somme in moneta britannica (pounds) comprese tra 10 sterline e 30 sterline a incrementi di 2 sterline. (Si supponga che 1 sterlina = 1,77$).

3. La relazione tra le scale dei gradi centigradi e Fahrenheit è espressa da questa formula:

   F = 1.8 ★ C + 32

   Scrivere un programma che stampi le equivalenti temperature in Fahrenheit delle temperature in gradi centigradi comprese tra 15 e 30°C, procedendo a intervalli di 1 °C.

   (SUGGERIMENTO: il corpo del programma potrebbe essere

   ```
   F = 1.8 ★ C + 32
   PRINT C,F
   ```

   Ciò è importante per la scelta del nome della variabile controllata).

Una volta scritti ed eseguiti tutti questi programmi, controllare le soluzioni a fronte di quelle nell'Appendice B.

Esperimento 7.3 completato	

La gente che ama la matematica talvolta viene confusa dal modo in cui viene usato il segno "=" in BASIC. Se ciò non si applica al lettore specifico, si può tranquillamente saltare questa sezione.
In matematica "="è usato nelle equazioni per asserire che due espressioni diverse hanno realmente lo stesso valore. L'equazione dice qualcosa che è vero. Per esempio, se l'insegnante di matematica scrive sulla lavagna

"$2x + 5 = 9$"

è possibile essere sicuri che per quel particolare x, che l'insegnante ha in mente, l'affermazione è corretta. Se così non fosse, sarebbe possibile immaginare la seguente conversazione:

L'allievo alza la mano.

Insegnante: Sì

Allievo: x è due

Insegnante: No. La risposta è 78

Allievo: Eh? Non capisco.

Insegnante: Ho detto una bugia scrivendo $2x + 5 = 9$!

In BASIC "=" è usato in due sensi diversi, nessuno dei quali corrisponde a quello usato in matematica.
In un comando LET, il segno significa "diventa". Si tratta di una *istruzione* per calcolare il valore dell'espressione alla destra e di inserire questo valore nella variabile sulla sinistra. Le istruzioni non sono affermazioni e non ha senso dire che sono o non sono vere. (Esse potrebbero essere sbagliate in un particolare contesto, ma ciò è un altro argomento). Il problema è che se viene trascurato LET, il comando *assomiglia* a un'equazione. Ciò non è assolutamente vero. Chiariamo questo punto:

in BASIC

Y=X+2

non informa il computer che Y è uguale a X+2 ma ordina di calcolare il valore di X+2 e di inserire il risultato nella variabile Y. Ecco alcuni punti da considerare:

- Q=Q+5 — è un comando BASIC utile e ragionevole
- P=Q e Q=P — non hanno gli stessi effetti
- X+1=5 — non è un comando BASIC lecito

In ciascun caso è possibile vedere il perchè? Cercare di spiegarlo con proprie parole.
L'altro uso di "=" è nelle condizioni. Si ricorderà che = è una delle sei possibilità di relazione tra quantità.

Gli esempi del suo uso sono

```
IF X=Y THEN 100

IF N$="YES" THEN 150
```

Di nuovo, non c'è alcuna implicazione che la condizione sia effettivamente vera; per contro, il comando è un ordine a procedere se la condizione è vera e a intraprendere una certa azione se lo è. Nelle condizioni "=" ha la stessa forza logica di qualsiasi altra relazione tipo < o >=. Vale però la pena di evitare la parola "uguale a" e chiamare il simbolo "è lo stesso di".

Per riassumere:

il BASIC usa "=" nei comandi LET, dove significa "diventa" e nelle condizioni dove significa "è lo stesso di", ma ciò che dice non è necessariamente vero. Capito?

Il programma di auto-test per questa unità è detto UNIT7QUIZ.

UNITA':8

A questo punto nel corso **64** inizieremo a scrivere i programmi. I primi sono corti e semplici. Successivamente, man mano che si approfondiscono conoscenza, esperienza e capacità, si potranno disegnare e scrivere programmi di sempre maggior complessità e interesse. La tabella dà qualche idea di che cosa si può fare.

Programma	*Numero di comandi*
Conversione Lire italiane in Lire sterline (Unità 7)	7
Programma quiz Unità 3	ca. 100
Programma per giocare a scacchi	ca. 5000
Programma per controllare un robot industriale	ca. 25000
Programma per gestire un sistema di prenotazioni aeree computerizzato	ca. 5000000

Naturalmente qualsiasi programma con più di circa 5000 comandi è sempre il risultato di uno sforzo di gruppo (occorrerebbe troppo tempo ad una sola persona per scriverlo) ma in ogni caso c'è sempre abbastanza spazio per il singolo programmatore.
Lavorando alla programmazione, ci si troverà spesso bloccati. Un programma per quanto scritto e immesso con grande cura non riesce a fare ciò che da esso ci si aspetta. Questa unità descrive alcuni dei modi in cui è possibile superare queste difficoltà. Occorre quindi leggerla e svolgere gli esercizi, ricordandosi che è sempre possibili tornare indietro quando (non *se*) ce n'è la necessità.
Quando si incontra la prima difficoltà di programmazione, si reagisce in maniera diversa. Alcuni si sentono insultati e furiosi; altri gettano la spugna e decidono che la programmazione non è cosa che li riguardi, alcuni sostengono che il programma "è giusto al 99,9%" e passano al successivo problema! Nessuna di queste reazioni è dettata dal buon senso. La sola cosa da fare è di trovare l'errore e di correggerlo. Può essere di grande conforto ricordare che ogni programmatore spesso si blocca e ciò vale anche per coloro che lavorano sui computer da 25 anni.
Gli errori di programma ricadono in tre gruppi. Il primo tipo, che è anche il più comune, è quello che si manifesta con SINTAX ERROR quando il computer tenta di eseguire un particolare comando. Ciò significa che il comando non segue le regole del linguaggio BASIC. Per esempio, potrebbe esserci un errore di ortografia in una parola chiave o potrebbero esserci troppe (o troppo poche) serie di virgolette. Gli errori di sintassi sono in gran parte provocati da errori di battitura e sono ovvii, dato che si vede dove sono; l'Appendice C dà in ogni caso un elenco di controllo del tipo di errore da consultare se si è in difficoltà. Il secondo tipo di errore di programma si ha quando il **64** trova un particolare comando impossibile da eseguire. Si supponga che la macchina arrivi al comando

130 GOTO 500

ma che non ci sia un comando contrassegnato 500. Ciò fa fermare la macchina e fa comparire un messaggio di errore:

UNDEFINED STATEMENT IN 130

Sfortunatamente i messaggi di errore tendono ad essere scritti nel gergo del programmatore anziché nella normale lingua inglese, ma sono comunque spiegati a fondo nell'Appendice C.
Il terzo tipo di errore di programma è il più difficile da trovare e da correggere. Esso non dà luogo a messaggi di errore, ma fa sì che il computer visualizzi la risposta sbagliata al problema o entri in un'iterazione senza addirittura visualizzare nulla. La prima e più ovvia cosa da fare è di fermare la macchina, listare (LIST) il programma ed esaminarlo accuratamente. Ciò solitamente aiuta ad individuare l'errore. In ogni caso si supponga che ciò non sia possibile; si immagini cioè di aver dedicato alcuni minuti ad esaminare ciascun comando senza aver trovato nulla di sbagliato.
In questi casi occorre un metodo più potente per esaminare il funzionamento del programma. Il metodo è detto "controllo di programma" e consiste nel mettersi al posto del computer: si parte all'inizio del programma e lo si svolge passo passo, comando per comando, fino a che non si riesce ad individuare la causa del problema. Occorre essere pazienti e metodici e soprattutto escludere per il momento la propria intelligenza e lavorare attraverso la serie di istruzioni come uno stupido robot, senza cercare di effettuare "tentativi plausibili", generalizzazioni o usare qualsiasi altro tipo di scorciatoia.
Per imitare il computer occorre per prima cosa avere un'idea del modo in cui funziona. Si supponga di poter in qualche modo "congelare" il **64** tra due comandi nel mezzo dell'esecuzione di un programma, aprirlo e osservarne l'interno. Si scoprirebbe che:*
Innanzitutto il programma stesso è caricato nella memoria più o meno nella forma in cui è stato originariamente battuto.
Secondo, le variabili che il programma ha usato fino a questo punto. Ciascuna variabile occupa un certo spazio in memoria ed ha un *valore*, che potrebbe essere un numero o una stringa.
Terzo, il computer ha tenuto nota della sua posizione nel programma. In qualche punto (in effetti in una variabile speciale detta "puntatore di pro-

Se si togliesse il coperchio del* ***64* *non si vedrebbero effettivamente queste cose ma solo pochi chip di silicio e altri componenti. Comunque gli appropriati strumenti elettronici, mostrerebbero certamente le varie situazioni sopra citate.*

gramma") ricorda il numero della label del successivo comando che si deve eseguire.
Proviamo ora a scongelare brevemente il computer, quanto basta per eseguire un comando. Il comando che la macchina sceglie sarà naturalmente quello ricordato dal puntatore di programma. Osservando in un tempo successivo, si troveranno certamente delle modifiche che dipendono dal comando che è stato appena eseguito. Ecco alcune delle possibilità.

(a) un comando PRINT farà comparire qualche cosa sullo schermo.

(b) un comando LET creerà una nuova variabile, se ne occorre una, e il **64** inserirà un nuovo valore. Sia PRINT che LET sposteranno il puntatore di programma al successivo comando in sequenza cosicchè, quando il computer viene fatto ripartire, "sa" quale comando eseguire successivamente.

(c) un GOTO non visualizzerà nulla nè altererà alcuna variabile ma semplicemente ripristinerà il puntatore di programma in modo che indichi il comando citato nella GOTO. Per esempio il comando

130 GOTO 270

inserirà 270 nel puntatore di programma.

(d) il comando IF funzionerà allo stesso modo, salvo che viene elaborata per prima cosa la condizione. Se è *vera* il contatore di programma viene impostato esattamente come in un GOTO. Se è *falsa*, il contatore di programma viene semplicemente spostato sulla label del successivo comando in sequenza.

Osservare: 120 IF X = 5 THEN 170
130 PRINT "NO"

Se X ha il valore 5, la condizione è vera e il contatore di programma viene modificato in 170. Per contro, se X ha qualche altro valore, il contatore di programma è semplicemente fatto avanzare a 130.

(e) il comando stop indica che il programma è terminato visualizzando un messaggio BREAK. Non c'è necessità di continuare il programma oltre a questo punto.

Per imitare accuratamente il computer, occorre poter vedere tutte queste parti chiaramente: il programma, le variabili, lo schermo video e il contatore di programma. Un buon metodo consiste nell'usare una "tabella di controllo del programma" che è possibile disegnare su un pezzo di carta. La si potrebbe disegnare così:

CONTATORE DI PROGRAMMA 10

VARIABILI

SCHERMO	*PROGRAMMA*
	10 A=5
	20 PRINT "ALPHA="; A
	30 A=A*3
	40 B=A+37
	50 PRINT "BETA="; B
	60 STOP

Il programma che si intende controllare viene scritto sulla destra e il valore iniziale del puntatore di programma — e cioè il numero della label del primo comando da eseguire — nella parte superiore. Assicurarsi che il programma sia una copia esatta di quello di cui si sta cercando il difetto: in caso contrario, il controllo rappresenta una perdita di tempo.
Ora, è possibile partire. Il contatore di programma dice "10" cosicchè occorre prendere e interpretare il comando contrassegnato "10". Esso dice A=5, cosicchè deve essere un comando LET. Osservare nella casella contrassegnata VARIABILI alla ricerca di una A. Dato che non c'è n'è, bisogna scrivere A, un due punti e il valore 5. Infine, spostare il puntatore di programma sul successivo comando in sequenza, tracciando una riga sul valore precedente.

CONTATORE DI PROGRAMMA ~~10~~ 20

VARIABILI A: 5

SCHERMO	*PROGRAMMA*
	10 A=5
	20 PRINT "ALPHA="; A
	30 A=A*3
	40 B=B+37
	50 PRINT "BETA="; B
	60 STOP

Il successivo comando è interpretato allo stesso modo. Occorre dimenticarsi lo "scopo" del programma o qualsiasi conoscenza che si potrebbe avere sulla sua sequenza e prendere il comando 20 soltanto in quanto il puntatore di programma dice di farlo. Il comando è un PRINT ed è possi-

bile scoprire che esso visualizza "ALPHA=5". Scrivere ciò nella parte SCHERMO e far avanzare il contatore di programma in modo da avere:

CONTATORE DI PROGRAMMA ~~10~~ ~~20~~ 30

VARIABILI A: 5

SCHERMO	PROGRAMMA
ALPHA = 5	10 A=5
	20 PRINT "ALPHA="; A
	30 A=A★3
	40 B=A+37
	50 PRINT "BETA="; B
	60 STOP

Il successivo comando dà un nuovo valore ad un'esistente variabile A. Occorre qui per prima cosa calcolare l'espressione A★3 usando il vecchio valore (5) ed annotarlo, barrando il vecchio valore come segue:

A: ~~5~~ 15

Il comando successivo crea una nuova variabile. Continuare il controllo fino a che non si raggiunge STOP. Il risultato finale è:

CONTATORE DI PROGRAMMA ~~10~~ ~~20~~ ~~30~~ ~~40~~ ~~50~~ 60

VARIABILI A: ~~5~~ 15 B: 52

SCHERMO	PROGRAMMA
ALPHA = 5	10 A=5
BETA = 52	20 PRINT "ALPHA="; A
BREAK IN 60	30 A=A★3
READY.	40 B=A+37
	50 PRINT "BETA="; B
	60 STOP

Il successivo esempio comporta una semplice iterazione:

```
10 P= 1
20 PRINT P; P★P★P
30 P=P + 1
40 IF P<4 THEN 20
50 STOP
```

Il controllo di questo programma per quanto riguarda la riga 30 è lineare:

CONTATORE DI PROGRAMMA ~~10~~ ~~20~~ ~~30~~ 40

VARIABILI P: ~~1~~ 2

SCHERMO	PROGRAMMA
1 1	10 P=1
	20 PRINT P; P★P★P
	30 P=P+1
	40 IF P<4 THEN 20
	50 STOP

Il successivo comando in 40 è un IF. Per imitare il computer, valutare la condizione in cui 4 è minore di P. Dato che il valore corrente di P è 2 (che è ciò che si dice nella sezione VARIABILI) e 2 è chiaramente minore di 4, la condizione è vera. Tutto ciò che occorre fare, pertanto, è inserire 20 come nuovo valore nel contatore di programma. Si ottiene:

CONTATORE DI PROGRAMMA ~~10~~ ~~20~~ ~~30~~ ~~40~~ 20

VARIABILI P: ~~1~~ 2

SCHERMO	PROGRAMMA
1 1	10 P=1
	20 PRINT P; P★P★P
	30 P=P+1
	40 IF P<4 THEN 20
	50 STOP

Il controllo continua in questo modo fino a che la condizione è falsa, e il programma raggiunge STOP. Il risultato finale è:

CONTATORE DI PROGRAMMA ~~10~~ ~~20~~ ~~30~~ ~~40~~ ~~20~~ ~~30~~ ~~40~~ ~~20~~ ~~30~~ ~~40~~ 50

VARIABILI P: ~~1~~ ~~2~~ ~~3~~ 4

SCHERMO	PROGRAMMA
1 1	10 P=1
2 8	20 PRINT P; P★P★P
3 27	30 P=P+1
BREAK IN 50	40 IF P<4 THEN 20
READY.	50 STOP

ESPERIMENTO 8·1

Fare ora pratica e controllare i seguenti programmi. Usare una matita e una gomma per cancellare eventuali errori.

(a)

CONTATORE DI PROGRAMMA 10

VARIABILI

SCHERMO	*PROGRAMMA*
	10 X=5 20 Y=7 30 Z=X+Y 40 W=Y−X 50 PRINT X;Y;Z;W 60 STOP

(b)

CONTATORE DI PROGRAMMA 10

VARIABILI

SCHERMO	*PROGRAMMA*
	10 Q=1 20 PRINT "LEI MI AMA" 30 PRINT "LEI NON MI AMA" 40 Q=Q+1 50 IF Q<3 THEN 30 60 STOP

Una volta completati questi due esperimenti, controllare le risposte a fronte di quelle indicate nell'Appendice B.

Esperimento 8.1. completato	

ESPERIMENTO 8·2

Come è possibile usare questo tipo di controllo per trovare gli errori? Tutto dipende dall'alternarsi tra uno stato di obbedienza passiva tipo robot e uno stato di intelligenza umana. Innanzitutto occorre trasformarsi in robot e controllare un comando esattamente come lo eseguirebbe il computer. Quindi bisogna ritornare a considerarsi una persona umana e chiedersi "è ciò che mi aspettavo?". In questo caso si esegue il controllo. In caso contrario, c'è una buona indicazione del perchè il programma non funziona.
Ecco un semplice esempio. Si supponga di aver scritto un programma per visualizzare la tabellina del 12. Lo schermo che ci si aspetta è

TABELLINA DEL DODICI

1 ★ 12 = 12

2 ★ 12 = 24

3 ★ 12 = 36

(fino a)

12 ★ 12 = 144

Il programma ha tutte le parti giuste: un'iterazione, un comando per visualizzare un titolo e un comando PRINT per visualizzare ciascuna riga della tabellina. Esso si presenterà nella forma:

```
10 PRINT "TABELLINA DEL DODICI"
20 P=1
30 P=P+1
40 IF P< 13 THEN 30
50 PRINT P;"★12=";P★12
60 STOP
```

Quando si esegue questop programma, i risultati sono leggermente sconcertanti. Tutto ciò che si ottiene è

TABELLINA DEL DODICI

13 ★ 12 = 156

BREAK IN 60

READY

Che non è certo quello che ci si aspettava! L'errore potrebbe essere perfettamente ovvio ma non è possibile pretendere d'individuarlo. Occorre iniziare il controllo e dopo alcune fasi si ottiene

CONTATORE DI PROGRAMMA

VARIABILI P:

SCHERMO	PROGRAMMA
TABELLINA DEL DODICI	10 PRINT "TABELLINA DEL DODICI"
	20 P=1
	30 P=P+1
	40 IF P< 13 THEN 30
	50 PRINT P; "★12="; P★12
	60 STOP

e improvvisamente ci si rende conto che il valore di P procede per conto suo fino a 12 senza che nulla venga visualizzato. È ora chiaro che il comando PRINT dovrebbe essere all'interno dell'iterazione e non all'esterno. Il posto giusto è tra i comandi 20 e 30. Il comando IF a sua volta deve essere cambiato per saltare indietro al PRINT. Una rapida correzione produce

```
10 PRINT "TABELLINA DEL DODICI"

20 P=1

25 PRINT P; "*12="; P*12

30 P=P+1

40 IF P < 13 THEN 25

60 STOP
```

Il controllo di programma è una tecnica estremamente utile se si ha la pazienza di eseguirla passo per passo. Se si tira ad indovinare su intere sezioni di programma, è possibile fare lo stesso errore compiuto nello scrivere il programma originariamente e il controllo non rivelerà alcun errore.
Ci sono alcuni casi nei quali il metodo di controllo sopra descritto non funziona ed occorre pertanto sapere quali sono:

- Se un programma è molto ampio, un controllo diretto richiederebbe troppo tempo. Metodi più appropriati saranno descritti più avanti nel corso, quando più probabilmente serviranno.
- Se non si crede di poter aver fatto un errore, il controllo non può essere di molto aiuto. La maggior parte delle persone quando scrivono l'ultima riga di un programma, hanno la profonda certezza che "stavolta è quella giusta". La sensazione deriva dalla consapevolezza di non aver fatto errori ed è estremamente ingannevole. È molto meglio dire "stavolta ho sbagliato. Troviamo gli errori". Ma occorre abbassare un po' l'orgoglio!
- Se sono stati interpretati erroneamente alcuni aspetti fondamentali del BASIC, anche un controllo sarà di scarso aiuto. Per prendere un semplice esempio, si immagini qualcuno che creda fermamente — ma erroneamente — che nel BASIC il segno "–" significhi "addizione". Egli scrive un programma come questo per sommare due numeri:

```
10 A=34

20 B=19

30 PRINT "A=";A

40 PRINT "B=";B

50 PRINT "A PIÙ B=";A-B

60 STOP
```

egli pensa che ciò significa "più"!

Quando il programma viene eseguito, visualizza

```
A = 34

B = 19

A PIÙ B = 15

BREAK IN 60

READY
```

che è chiaramente sbagliato. Per contro, quando lo si controlla, si trova che il comando 50 dà

```
A PIÙ B = 52
```

che è ciò che si aspetta. Fintantochè si crede che "–" significhi "sommare", non si troverà mai l'errore!

Naturalmente la maggior parte delle incomprensioni sono molto più sottili di questa. Ciononostante se il controllo risulta diverso dal risultato visualizzato dal **64** e continua a rimanere tale quando si ripete il controllo, c'è una chiara dimostrazione che c'è qualcosa che non si è compreso perfettamente in materia di tecnica di programmazione. Se è possibile, farsi consigliare da qualcuno che conosca a fondo il BASIC*; altrimenti tornare all'inizio di questo manuale e controllare ogni argomento. Quasi sempre ciò porterà alla luce il problema.
Talvolta — sebbene molto raramente la difficoltà potrebbe essere provocata da un guasto meccanico nel computer. Le macchine moderne tipo il **64** sono estremamente robuste ed affidabili e quando si rompono la cosa è solitamente ovvia: il cursore non compare quando si accende l'apparecchio o risulta impossibile caricare i programmi dal registratore a cassetta. In pratica non occorre mai incolpare il computer per la mancata esecuzione del programma fino a che non si è esaminata due o tre volte qualsiasi altra possibilità. Quando si invia la macchina per la riparazione, occorre spiegare esattamente perchè si pensa che sia rotta ed allegare una copia del programma che non viene eseguito correttamente.

**Ci sono numerose persone che conoscono il BASIC. Non occorre conoscerle personalmente: basterà un'inserzione nel negozio locale per trovare rapidamente aiuto.*

Ecco due programmi contenenti errori da trovare e da correggere.

(a) Si supponga che questo programma visualizzi una tabella di conversione dei galloni in litri, iniziando con un gallone e terminando con 10 galloni (1 gallone = 4.5 litri)

```
10 PRINT "GALLONI","LITRI"
20 G=1
30 PRINT G, 4.5*G
40 G=G+1
50 IF G > 11 THEN 30
60 STOP
```

(b) Questo programma dovrebbe rappresentare una soluzione al problema 1 nell'Unità 7 per visualizzare un triangolo. È stato scritto da qualcuno che stava imparando il BASIC:

```
10 A$ = "*"
20 PRINT A$
30 A$ = "**"
40 IF A$<>"*********" THEN 20
50 STOP
```

Esperimento 8.2 completato	

ESPERIMENTO 8·3

Si supponga che il programma sul nastro UNIT-8PROG debba visualizzare la tabellina del 7, ma contenga parecchi errori. Caricarlo, trovare e correggere gli errori. Controllare le risposte nell'Appendice B.

Esperimento 8.3 completato	

UNITA':9

ESPERIMENTO 9·1

Disegneremo ora qualche altra figura ma stavolta faremo in modo che sia il **64** a fare tutto il lavoro al nostro posto. Se si ripensa alle Unità 2 e 3, si ricorderà che per disegnare sullo schermo è possibile usare un certo numero di funzioni di controllo:

- Movimento del cursore in quattro diverse direzioni
- Scelta di otto diversi colori
- Inversione del colore e dello sfondo
- Spostamento del cursore al punto di partenza nell'angolo superiore sinistro
- Cancellazione dello schermo

Queste funzioni usano in comune dei tasti sulla tastiera, cosicchè occorre spesso dover usare i tasti SHIFT o CTRL o C= per scegliere le funzioni che di volta in volta occorrono.

Non si sarà certo dimenticato che è possibile impostare il colore del margine e del fondo usando un'istruzione POKE e un numero di codice ricavato dalla tabella di pagina 19.
Il **64** può anche eseguire disegni sullo schermo sotto controllo di un programma. Ogni programma ha l'uso di tutte le funzioni di controllo dello schermo: può scegliere qualsiasi colore per i suoi caratteri e può cancellare lo schermo ogni volta che è necessario e può spostare il proprio cursore (che è invisibile all'operatore) su qualsiasi posizione usando le funzioni di controllo del cursore.
Naturalmente il **64** esegue queste cose soltanto obbedendo ai comandi che gli sono stati impartiti. Inserire le funzioni di controllo in un comando è facile: basta includerle in stringhe insieme ad altri caratteri da visualizzare. È possibile all'inizio trovare la cosa piuttosto complicata. Ma se si batte una stringa e vi si include una funzione di cancellazione dello schermo, l'intero schermo non scompare forse quando la si batte? In effetti ciò non si verifica, come si vedrà dal successivo esperimento.

Si ricorda che nell'Unità 2 si diceva di "non usare le doppie virgolette, perchè danno effetti strani". Ora si cercherà di scoprire quale effetto esse realmente hanno e perchè sono così utili.
Quando si inizia a battere un comando (ad esempio dopo un READY o un RETURN), il **64** è nel modo "normale". Le funzioni di controllo tipo la selezione dei colori o il movimento del cursore funzionano nel modo previsto. Non appena si batte un carattere di virgolette per contrassegnare l'inizio di una stringa, la macchina passa al modo "virgolette". I normali caratteri tipo lettere e segni grafici sono sempre trattati nel modo normale ma non vengono più eseguite le funzioni di controllo: per contro, vengono inseriti nella stringa come caratteri "speciali", prevalentemente lettere, segni o grafici su un fondo in negativo. La macchina ritorna al modo normale quando si battono una seconda volta le virgolette (per terminare la stringa) oppure se si impartisce il comando RETURN.
Accendere il **64,** battere un segno di virgolette seguito da tutte le funzioni di controllo, una per una.

Funzione	*Tasto*	*Simbolo visualizzato*
Cancellazione schermo	SHIFT e CLR HOME	
Cursore in posizione di partenza	CLR HOME	
Cursore verso l'alto	SHIFT e CRSR	
Cursore verso il basso	CRSR	
Cursore a sinistra	SHIFT e CRSR	
Cursore a destra	CRSR	
Nero	CTRL e BLK	
Bianco	CTRL e WHT	
Rosso	CTRL e RED	
Celeste	CTRL e CYN	
Porpora	CTRL e PUR	
Verde	CTRL e GRN	
Blu	CTRL e BLU	
Giallo	CTRL e YEL	
Arancio	C= e 1	
Marrone	C= e 2	
Rosso chiaro	C= e 3	
Grigio 1	C= e 4	
Grigio 2	C= e 5	
Verde chiaro	C= e 6	
Blu chiaro	C= e 7	
Grigio 3	C= e 8	
Negativo attivato	CTRL e RVS ON	
Negativo disattivato	CTRL e RVS OFF	

Proviamo ora a vedere in azione qualcuno di questi comandi. Innanzitutto assicurarsi che il televisore sia correttamente regolato per il colore, usando il programma TESTCARD se necessario.

Successivamente far sì che il computer visualizzi la parola "EDINBURGH" in giallo. Battere il comando

Ciò che effettivamente compare sullo schermo (sempre in blu) è

PRINT" π EDINBURGH". Il simbolo π

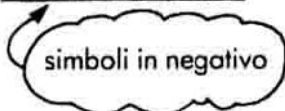
(in negativo)

in negativo è il codice di "giallo".

Battere ora il tasto RETURN. Sullo schermo compare la parola EDINBURGH, in giallo.
Questo esperimento illustra il principio abbastanza chiaramente: quando viene battuta una funzione di controllo all'interno di una stringa, questa viene posta in effetto nel momento in cui viene battuta ma eseguita soltanto quando la stringa deve essere visualizzata dal computer. Si vedrà che il cursore lampeggiante è rimasto giallo. Portarlo in nero o blu, a seconda della preferenza, battendo la corretta funzione di controllo — senza virgolette.
Un comando PRINT che ha per effetto un cambio di colore può essere reso parte di un programma, esattamente come qualsiasi altro comando. Impostare ed eseguire quanto segue:

e YEL ANDREWS"

40 STOP
Il comando 30 mostra che è possibile inserire più di una funzione di controllo in una stringa.
Nelle stringhe possono anche essere inserite le funzioni di controllo dello schermo e del cursore. Battere quanto segue:

Sullo schermo ciò appare come

Quando si batte RETURN, le funzioni di controllo sono eseguite effettivamente. Lo schermo viene cancellato e il cursore viene spostato di tre posti verso il basso e di tre verso destra mentre la parola PARIS compare in rosso a metà al centro dello schermo. Provare da soli.
In generale è possibile far sì che il **64** disegni parole e simboli ovunque, includendo in una stringa il numero appropriato di spostamenti del cursore.
Ogni volta che si fa in modo che il computer disegni qualche cosa sullo schermo, non si vuole ovviamente cancellare tutto visualizzando READY e il cursore lampeggiante. Per superare questa difficoltà basta usare un "arresto dell'iterazione" o un GOTO che salta su se stesso. Una volta che il computer raggiunge questo comando, inizia a dare la caccia alla propria coda e non visualizzerà READY fino a che qualcuno non batte il tasto RUN STOP. Questo programma per esempio, visualizzerà LONDON in bianco al centro di uno schermo nero:

30 GOTO 30

Impostare questo programma, eseguirlo e quindi interromperlo con il tasto RUN STOP. Lo schermo sarà ancora nero e il cursore sarà bianco ma è possibile rapidamente tornare alla normale condizione tenendo abbassato RUN STOP e premendo contemporaneamente RESTORE. In effetti, è sempre possibile far ciò se la macchina si blocca per qualsiasi motivo: è meglio che spegnerla e riaccenderla, in quanto in quest'ultimo caso il programma andrebbe perso.

Come breve esercizio, fare in modo che il **64** visualizzi sullo schermo parole e profili di colori diversi in varie posizioni. Ricordarsi che la funzione SHIFT e CLR HOME cancella lo schermo cosicchè se il programma ha una sequenza d'istruzioni PRINT, soltanto la prima deve iniziare con questa funzione — quantunque alcune delle altre potrebbero benissimo partire con CLR HOME.

Esperimento 9.1 completato	

Tutti sanno che i moderni orologi e sveglie sono controllati da cristalli di quarzo che sono estremamente precisi su lunghi periodi di tempo. Anche il **64** incorpora un cristallo al quarzo che vibra parecchi milioni di volte ogni secondo e viene usato — tra gli altri scopi — come orologio interno di controllo. Questo orologio non ha un proprio quadrante. Per contro è trattato esattamente come una variabile stringa, cosicchè è possibile visualizzare l'ora sullo schermo ogniqualvolta occorre. In nome della variabile di orologio è TI$.
Quando si fa partire per la prima volta qualsiasi orologio, occorre mettere a punto l'ora. Anche il **64** non fa eccezione. È possibile regolare l'orologio dalla tastiera, battendo un comando tipo

TI$ = "193746" RETURN

che predispone l'orologio alle ore 19.37 minuti e 46 secondi.
Se si vuole impostare l'ora molto accuratamente, è meglio attendere — ad esempio — il segnale orario delle 9 alla radio. Immediatamente prima del segnale orario, battere

TI$ = "090000"

quindi, battere RETURN non appena si ode l'ultimo "beep" del segnale orario.
Una volta che l'orologio del **64** è stato regolato, esso manterrà il tempo con uno scarto di pochi secondi al giorno, fino a che la macchina non viene spenta. Non c'è necessità di ripristinarlo o di cambiarlo dall'interno di un programma.
Per visualizzare l'ora basta citare TI$ in un comando PRINT.
Ora provare a impostare l'orologio interno del **64** usando il proprio orologio (non ha importanza se la regolazione non è molto accurata). Quindi visualizzare il valore di TI$ parecchie volte, usando un comando PRINT. Vedere come il valore dei secondi cambia da una volta alla successiva.
Visualizzare ora in continuazione l'ora, eseguendo il programma

```
10 PRINT TI$
20 GOTO 10
```

Interrompere questo programma, attendere alcuni minuti e farlo ripartire. Si vedrà che l'ora è sempre corretta e che l'orologio ha continuato a marciare.
Questo metodo per visualizzare l'ora non è interessante. È invece possibile far sì che il **64** si comporti come un rispettabile orologio digitale mediante un programma del tipo che segue:

comando 10: Scegliere una griglia color porpora con sfondo color giallo

comando 20: Cancellare lo schermo

comando 30: Spostare il cursore in posizione di partenza, quindi verso il basso di 9 righe e verso destra di 6 spazi; non occorre una nuova riga

comando 40: Visualizzare TI$

comando 50: Saltare indietro al comando 30

Scrivere il codice per questo programma nella casella che segue: quindi impostarlo sulla tastiera del **64** e provarlo. Se *veramente* non si riesce, osservare la versione corretta nell'Appendice B ma non procedere fino a che non si è ristudiato il tutto accuratamente e trovato come funziona.

Esperimento 9.2 completato	

ESPERIMENTO 9·3

Le iterazioni controllate sono spesso utili per disegnare profili sullo schermo. Si supponga di voler disegnare un blocco di 10x10 punti rossi nell'angolo superiore sinistro. Ciò può essere effettuato visualizzando 10 righe, ciascuna con 10 segni grafici tipo ●:

10 PRINT " SHIFT e CLR HOME";

20 J=1

30 PRINT " CTRL e RED ●●●●● ●●●●●"

40 J=J + 1

50 IF J<11 THEN 30

60 GOTO 60

Questo programma riunisce parecchie delle idee finora incontrate nelle unità precedenti. Il punto e virgola al termine del comando 10 impedisce che la macchina inizi una nuova riga dopo aver cancellato lo schermo, cosicchè la prima riga di punti rossi compare nella parte superiore. Le istruzioni dal 20 al 50 formano un'iterazione controllata e 60 è un arresto dell'iterazione.
Immettere il programma ed eseguirlo così come sta. Quindi, interromperlo e provare da soli gli effetti

(a) della rimozione del punto e virgola dopo "CLR HOME"

(b) del cambiamento di 11 nel comando 50 in qualche altro valore (ed esempio 15)

(c) della rimozione del comando 60

È possibile naturalmente eseguire queste modifiche listando (LIST) e correggendo. Ricordarsi in ogni caso di riportare il colore del cursore al blu o al nero prima di iniziare!
Per ottenere un blocco pieno di colore, si usano gli spazi in negativo. Provare a cambiare la riga 30 nella seguente

PRINT " CTRL e RED CTRL e RVS ON ← 10 spazi →"

ed eseguire il programma.
Cosa succede se si vuole avere più di un blocco di colore nella stessa immagine? Il trucco consiste nello spostare il cursore della macchina alla prima riga dell'area e quindi riempirla senza interferire con il colore che è già sullo schermo. Si osservano ora due esempi:

(a) Per dipingere un blocco 10x10 blu immediatamente al disotto di quello rosso:
La metà inferiore dello schermo è vuota cosicchè non occorre preoccuparsi di danneggiare un altro colore già presente. Inoltre, dopo aver disegnato il blocco rosso, il cursore si troverà nella posizione adatta.
È possibile estendere il programma aggiungendo

60 J=1

70 PRINT " CTRL e BLU CTRL e RVS ON ← 10 spazi →"

80 J=J + 1

90 IF J< 11 THEN 70

100 GOTO 100

Notare che l'arresto dell'iterazione è stato spostato al termine del programma al quale appartiene. J è usato come variabile di controllo nelle iterazioni del rosso e del blu: ciò è perfettamente giusto in quanto il blocco rosso è completamente finito prima che inizi quello blu e a J non è stato chiesto di eseguire due lavori contemporaneamente.

(a) Per dipingere un blocco nero 10x10 di fianco a quello rosso.

La riga di partenza è quella superiore cosicchè nel disegnare l'area nera occorre fare attenzione a non danneggiare il blocco rosso che è già presente. Ciò può essere fatto spostando il cursore al punto di partenza e visualizzando 10 righe, ciascuna delle quali inizia con 10 movimenti di "cursore a destra" per saltare la parte rossa. L'estensione di programma è

130 J=J + 1

140 IF J < 11 THEN 120

150 GOTO 150

Ora assemblare questo programma, batterlo e provarlo. Notare che ci sono tre iterazioni separate che vengono eseguite una dopo l'altra.
Provare ad estendere il programma per inserire un blocco porpora sotto quello nero...
Come esercizio finale, cercare di scrivere programmi per visualizzare qualche semplice bandierina o altri profili che riempiano l'intero schermo. Occorre però fare molta attenzione perchè sul percorso ci sono molte trappole in attesa.

- Il normale significato di un punto e virgola al termine di un comando PRINT è "non iniziare una nuova riga". Se al **64** viene ordinato di inserire un carattere nella posizione più a destra di una riga, esso *automaticamente* sposta il cursore ad una nuova riga. Gli schermi che si vuole presentino righe complete devono pertanto essere seguiti da punti e virgola a meno che non si desideri effettivamente la presenza di una riga vuota.

- Non c'è modo di usare un comando PRINT per scrivere un carattere nell'angolo inferiore destro dello schermo senza far spostare l'intero schermo verso l'alto. Il modo per ottenere questo quadratino nel colore adatto è di scegliere opportunamente l'intero colore dello sfondo.

Occorre pianificare il disegno accuratamente usando carta quadrettata come guida. Quando è il momento di scrivere programmi, bisogna essere preparati a compiere errori ed a non arrabbiarsi se occorrono parecchi tentativi per ottenere il risultato giusto. Ricordarsi che s'impara attraverso il successo — e non attraverso l'insuccesso — per cui non rinunciare!
Per iniziare, ecco un programma per la bandiera francese.

blu	bianco	rosso

Si farà la striscia centrale larga 14 caratteri e le due laterali 13 caratteri ciascuna. 13+14+13=40.
Gli appropriati colori di partenza sono un fondo rosso e un riquadro nero.
È possibile costruire la bandiera visualizzando 25 righe, ciascuna con 13 quadratini bianchi e 14 blu. Ricordarsi che l'ultimo deve essere diverso in quanto non deve essere seguito da una nuova riga. È possibile inserire le prime 24 righe in un'iterazione controllata ma l'ultima richiederà un comando a sè.

Arriviamo ad avere

Eseguire questo programma e studiarlo accuratamente fino a che non se ne comprende ogni simbolo. Cercare ora di disegnare proprie bandiere, evitando però, per il momento, quelle con elementi diagonali. Provare con la bandiera dell'Islanda che è illustrata a pagina 21. È possibile controllare la risposta con quella indicata nell'Appendice B.

Il programma di auto-test per questa unità è detto UNIT9QUIZ.

UNITA':10

Dalle precedenti unità ci siamo fatti un'idea che i comandi possono essere scritti una volta sola, ma eseguiti ripetutamente parecchie volte. Ciò si verifica ogniqualvolta che in un comando si inserisce una iterazione.
Su scala molto più ampia, una cosa simile avviene con i programmi completi. La maggior parte dei programmi è studiata per essere utile, il che significa che essi sono memorizzati e distribuiti su nastri o su ROM e usati molte volte da persone diverse. A titolo di esempio, osservare i vari programmi registrati su nastro che formano parte di questo corso.
Iniziamo considerando tutti i programmi che sono stati finora scritti personalmente. Ognuno di essi ha l'inconveniente di produrre sempre lo stesso risultato ogni volta che viene eseguito. Difficilmente un programma del genere sarebbe molto utile!
Per fare un esempio specifico, torniamo indietro al programma che calcola e visualizza una tabella di conversione tra le lire sterline e le lire italiane. Il programma era così strutturato:

```
10 PRINT"£","LIRE"

20 PRINT

30 PS=5

40 PRINT PS,2350★PS

50 PS=PS+5

60 IF PS<80 THEN 40

70 STOP
```

Il giorno in cui è stata scritta l'Unità interessata, il tasso di cambio era realmente 2350 lire per sterlina cosicchè il programma avrebbe dato risultati corretti. Il tasso di cambio è poi sceso a 2175. Qualsiasi banca che usasse questo programma originale per vendere lire in cambio di sterline, accumulerebbe delle grosse perdite.
Come si possono migliorare le cose? Un programmatore avrebbe un approccio ovvio consistente nel variare la riga 40 in modo da farla apparire

```
40 PRINT PS,2175★PS
```

Sfortunatamente questa idea non porterebbe molto lontano. La maggior parte di coloro che usano il computer non sono programmatori o se lo sono, non sono interessati alle particolarità di questo genere del programma!
Per rendere i programmi più flessibili, più adattabili alle esigenze quotidiane, occorre una nuova funzione: una funzione che consenta all'utente di fornire informazioni che il programmatore non poteva conoscere nel momento in cui il programma è stato scritto.
Un programma che serve a parecchi gruppi di persone diverse e consenta a ciascuna di essere di risolvere una propria particolare versione di un problema. Per esempio, si supponga che il programma di conversione delle monete consenta all'utente di indicare il tasso corrente di cambio ogni volta che viene usato; in tal caso diventerebbe immediatamente utile a tutte le banche di tutto il mondo e funzionerebbe correttamente per qualsiasi tasso di cambio immaginabile.
Si supponga di disegnare un programma che debba essere usato da altri. Si inizia decidendo quali quantità occorre lasciare indefinite e che il programma chiederà all'utente di fornire. Nell'esempio, il tasso di scambio è chiaramente una tale quantità: esso deve essere sconosciuto al programmatore ma noto all'utente. Si assegnano le quantità incognite alle variabili e si attribuiscono loro dei nomi di conseguenza. Per esempio, un nome adatto per il tasso di cambio potrebbe essere RE. È possibile quindi scrivere un programma usando nomi simbolici invece dei valori effettivi (che non possono essere noti in anticipo). Pertanto la riga 40 del programma di cambio potrebbe comparire come segue:

```
40 PRINT PS, RE★PS
```

Naturalmente manca qualcosa a questa descrizione. Non si possono conoscere i valori delle variabili ma la macchina deve conoscerli quando esegue il programma. Il comando che consente all'utente di inserire le informazioni mancanti ha la parola chiave INPUT. Questa è seguita dal nome (o dai nomi) delle variabili necessarie. Quando il comando INPUT viene eseguito, esso attende che l'utente batta un valore, che quindi memorizza nella variabile specificata. Ora può esere eseguito il resto del programma che usa questa variabile.
Prima di dare un esempio, è bene sottolineare un punto estremamente importante: ogni programma con un comando INPUT deve dire all'utente esattamente cosa gli si richiede. Ciò può essere solitamente fatto con le istruzioni PRINT.

ESPERIMENTO 10·1

Studiare accuratamente il seguente programma:

```
3 PRINT "STAMPA TASSO"
4 PRINT "CAMBIO ODIERNO"
5 PRINT "TRA £ E LIRE"
6 INPUT RE
10 PRINT "£","LIRE"
20 PRINT
30 PS=5
40 PRINT PS, RE*PS
50 PS=PS+5
60 IF PS<80 THEN 40
70 STOP
```

Notare come il programma non presume qualsiasi particolare tasso di cambio, ma usa la variabile RE per rappresentarlo ogniqualvolta occorre. Il programma inizia dicendo all'utente ciò che occorre e chiedendogli di fornire un valore.
Immettere il programma, controllarlo attentamente e battere RUN. Si immagini ora di essere un utente: un cambiavalute che non conosce nulla di programmazione. Sullo schermo la macchina sta chiedendo di battere qualcosa per cui occorre inserire l'appropriata cifra e quindi battere RETURN
Non appena si è fatto, lo schermo si riempie con una tabella di conversione che consente di iniziare a lavorare.
Eseguire il programma parecchie volte e notare come può gestire diversi tassi di cambio. Anche se la lira venisse rivalutata ad un livello di 23.7, rispetto alla sterlina, il programma produrrebbe sempre risposte esatte.
Ora è il momento di riprendere la propria personalità originale di programmatore. Quando l'intero programma era in funzione, la visualizzazione del cursore e l'attesa della battitura di qualcosa da parte dell'utente rappresentava in effetti l'esecuzione del comando INPUT.
Il comando INPUT compare in parecchie forme leggermente diverse. Ne osserveremo qulche esempio e citeremo qualche regola generale.

1. Inizializzare il **64** battendo NEW e battere quindi

```
10 PRINT "DIMMI IL TUO NOME"
20 INPUT N$
30 PRINT "HELLO SPACE";N$;"!"
```

Eseguire questo programma e vedere cosa succede. L'esempio mostra come funziona il comando INPUT con stringhe nonchè numeri. È possibile usare questa sequenza — o qualcuna analoga — pressochè all'inizio di qualsiasi programma se si vuole che il computer si comporti in modo "amico" per l'utente. Se il programma fosse un quiz di qualsiasi tipo, si sarebbe potuto usare il valore di N$ in un comando tipo

```
40 PRINT "NO SPACE"; N$;".PUOI
   FARE DI MEGLIO"
```

(In caso di dubbio su ciò che visualizza questo comando, basta aggiungerlo all'estremità del programma che è già nel **64** ed eseguire il programma nuovo).

2. Provare

```
10 INPUT "NOME";N$
20 PRINT "CIAO SPACE"; N$
```

Questo esempio mostra come è possibile includere nel comando INPUT un breve pezzo di informazione descrittiva. L'informazione compare sullo schermo come una guida per l'utente, immediatamente prima del punto di domanda.
Il comando 10 nell'esempio è uquivalente alla sequenza

```
PRINT "NOME";
INPUT N$
```

Notare che la stringa di parole descrittive deve contenere meno di 70 caratteri e deve essere seguita da un punto e virgola.

3. Provare infine

```
10 PRINT "DATI DUE NUMERI DA
   SOMMARE"
20 INPUT A,B
30 PRINT "SOMMA=";A+B
40 STOP
```

Il comando INPUT aspetta ora due valori e l'utente deve batterli separati da una virgola o premendo il tasto RETURN.
(Cioè potrebbe battere

ad esempio 43,19

oppure { 43
19

In generale, il comando INPUT può chiedere all'utente qualsiasi numero di variabili ma è meglio tenere il numero basso, limitandolo a due per evitare confusione. Nel comando stesso, i numeri delle variabili sono separati da virgole. Dopo aver eseguito questo programma alcune volte, s'immagini di essere un utente alquanto stupido e provare a battere qualcosa che non abbia senso — ad esempio

DONALD,DUCK

In computer accetterà qualsiasi cosa come una stringa ma se tenta di immettere un numero e gli viene dato qualcosa che non può essere un numero, visualizza il messaggio

REDO FROM START

e dà un'altra possibilità di ritentare.
Talvolta si vuole interrompere un programma quando sta eseguendo un comando INPUT e visualizzando il cursore. In questo condizioni, il tasto RUN STOP è disabilitato. Occorre in tal caso tenerlo abbassato e battere contemporaneamente il tasto RESTORE alla destra della tastiera principale. Provare!

Esperimento 10.1 completato	

ESPERIMENTO 10·2

Scrivere programmi utili è facile, posto di ricordarsi che il programmatore e l'utente sono due persone diverse. Si supponga che l'utente non conosca la programmazione (cosicchè non ci si può aspettare che esegua una lista — LIST — del programma per trovare ciò che fa). In generale il programmatore non può "parlare" all'utente se non facendo in modo che il **64** visualizzi il messaggio sullo schermo e l'utente non può rivolgersi al programmatore per nulla, cosicchè è bene che il programma non lasci questioni in sospeso.
Quando si disegna un programma, si immagini di essere una mosca sulla parete in osservazione di qualcuno che tenta di usarlo. Provarsi a immaginare tutto ciò che si potrebbe fare di sbagliato cercando di impedirglielo e facendo in modo che il programma fornisca un'esauriente guida.
Quando il programma è scritto, è possibile provarlo mettendosi al posto dell'utente; successivamente, come prova finale, occorrerebbe pregare un amico o un parente di fare da cavia facendogli provare il programma a sua volta. Se la cavia deve porre domande su cosa fare o sul significato delle risposte, significa che la prova non è riuscita e che occorre ridisegnare il programma.

Scrivere programmi che svolgono i seguenti lavori:

(a) Visualizzare qualsiasi tabellina di moltiplicazione scelta dall'utente.

(b) Chiedere all'utente (che si presume essere un uomo sposato) il cognome e quindi il nome della moglie, quindi visualizzare il nome completo della moglie.

Le soluzioni sono indicate nell'Appendice B, ma non consultarle fino a che non si è fatto tutto il possibile per scrivere questi programmi da soli.

Esperimento 10.2 completo	

Il quiz per questa Unità è detto UNIT10QUIZ.

UNITA':11

Una delle caratteristiche più interessanti della programmazione è la sua ricchezza e varietà. Lo stesso computer, se opportunamente programmato, può servire da calcolatore, da macchina per insegnamento, da strumento musicale, da monitor per sorvegliare un paziente in ospedale o per fare pressochè qualsiasi altra cosa utile cui è possibile pensare. Questa potenza deriva dall'enorme numero di modi in cui è possibile combinare i pochi comandi base.
Finora il nostro vocabolario totale di comandi usato all'interno dei programmi era costituito da sette elementi:

PRINT, LET, GOTO, IF, INPUT, STOP e POKE

Naturalmente ci sono altri comandi BASIC che occorrerà imparare, ma in questa unità esploreremo il potenziale dei comandi che già si conoscono. Il comando più flessibile di tutti è IF. Nelle precedenti unità è stato usato per controllare le iterazioni ma è anche utile in molti altri casi. Per esempio, può provare dati o elementi di informazioni forniti dall'utente in modo da far compiere al computer il tipo di azione appropriata.

Si immagini di voler organizzare un'agenzia matrimoniale computerizzata la cui prima funzione consista nell'approntare un programma per indicare le età di cui i clienti dovrebbero tener conto nella scelta del partner. Per tradizione, un uomo dovrebbe sposare una ragazza con un'età pari alla metà della sua più sette. E cioè implica, ovviamente, che una ragazza che deve cercare un marito con il doppio della sua età meno 14.
Chiaramente, il programma deve iniziare chiedendo l'età del cliente. Quindi, per dargli il consiglio giusto, deve scoprire se il cliente è un uomo o una donna. Il programma verrà usato da uomini e donne, cosicchè deve comprendere un gruppo di comandi separati per fornire consigli a ciascuno dei due sessi. Infine, deve esserci un comando IF per scegliere il gruppo effettivamente necessario in una particolare occasione. Qui di seguito è indicata la prima versione del programma. Studiarla accuratamente e spiegare esattamente perchè è incluso ciascun comando:

```
10 INPUT "QUANTI ANNI HAI";AG

20 INPUT "MASCHIO O FEMMINA";SX$

30 IF SX$="MASCHIO" THEN 70

40 PRINT "DEVI CERCARE"

50 PRINT "UN UOMO DI";2★AG−14; "ANNI"

60 STOP

70 PRINT "DEVI TROVARE"

80 PRINT"UNA RAGAZZA DI";AG/2+7; "ANNI"

90 STOP
```

Si sarà notato l'uso della variabile AG per contenere l'età del cliente e SX$ il suo sesso. La condizione SX$="MASCHIO" è vera se il cliente risponde (maschio) alla domanda "MASCHIO O FEMMINA?" (maschio o femmina). L'espressione AG/2+7 è il modo con cui il basic dice "metà dell'età + 7" e 2★AG−14 significa "due volte l'età meno 14". Una volta osservato il programma, provare a *prevedere* il più accuratamente possibile cosa comparirà sullo schermo:

(a) per un uomo di 20 anni e (b) per una ragazza di 22. Usare i riquadri che seguono. Il primo riquadro è parzialmente compilato.

Immettere il programma nel **64.** Provarlo usando vari tipi di clienti e controllare che entrambe le previsioni siano corrette.
Questo semplice esempio mostra che la azione del computer non deve essere fissata in anticipo dal programmatore ma può essere resa dipendente dalle informazioni fornite dall'utente.
I programmi devono spesso prendere complicate serie di decisioni cosicchè per pianificarli dobbiamo usare un tipo speciale di diagramma detto schema di flusso. Lo schema di flusso per un programma di questo genere è il seguente:

INPUT ETÀ

INPUT SESSO

SESSO = "MASCHIO"

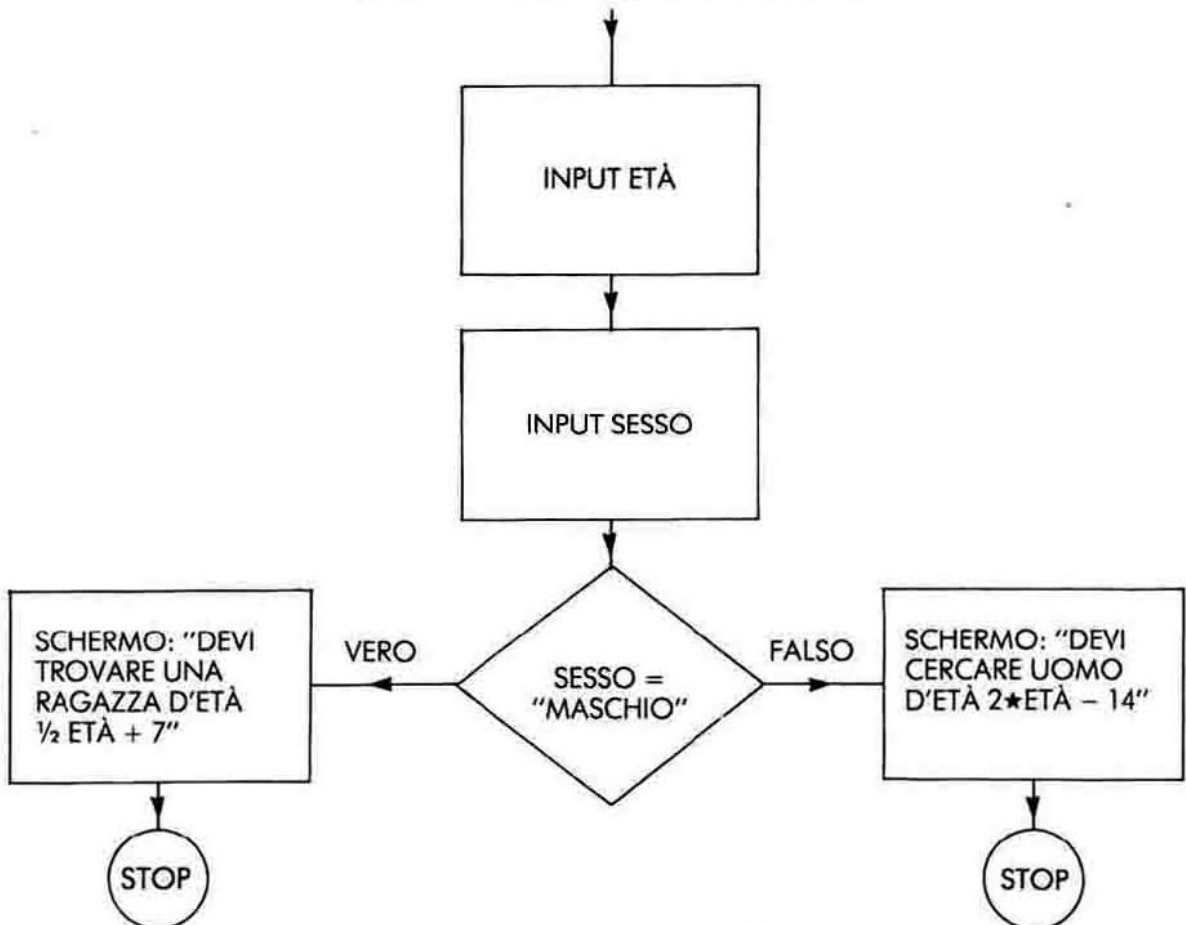
VERO

SCHERMO: "DEVI TROVARE UNA RAGAZZA D'ETÀ ½ ETÀ + 7"

STOP

FALSO

SCHERMO: "DEVI CERCARE UOMO D'ETÀ 2★ETÀ – 14"

STOP

Uno schema di flusso si compone di un certo numero di blocchi collegati da righe con frecce. Ci sono quattro tipi di blocchi:

(a) Un quadrato o un rettangolo contiene la descrizione di un'azione semplice che può essere successivamente tradotta in uno o due comandi BASIC. Nello schema di flusso esemplificativo i due rettangoli superiori sono esempi di questo tipo. Le righe con frecce mostrano che il programma inizia eseguendo il primo blocco quindi va al secondo nell'ordine.

(b) Un rombo contiene una condizione che può essere vera o falsa. Nel rombo entra una linea ma ne escono sempre due, contrassegnate VERO o FALSO (o talvolta SI e NO). Il rombo corrisponde ad un comando IF ed istruisce il computer a provare la condizione e a seguire la linea VERO o FALSO a seconda del risultato.

(c) La figura terminale, che dice al computer di interrompere l'esecuzione del programma. È un piccolo cerchio con la parola STOP.

(d) La nuvoletta (che non compare nell'esempio). Si tratta di un simbolo per un'azione che è troppo complicata per poterla scrivere in dettaglio. Solitamente la nuvoletta potrebbe essere ampliata in un altro schema di flusso completo, esattamente come si farebbe con una mappa stradale nazionale che viene affiancata da mappe dettagliate delle diverse città.

Uno schema di flusso è realmente una "mappa" del programma.

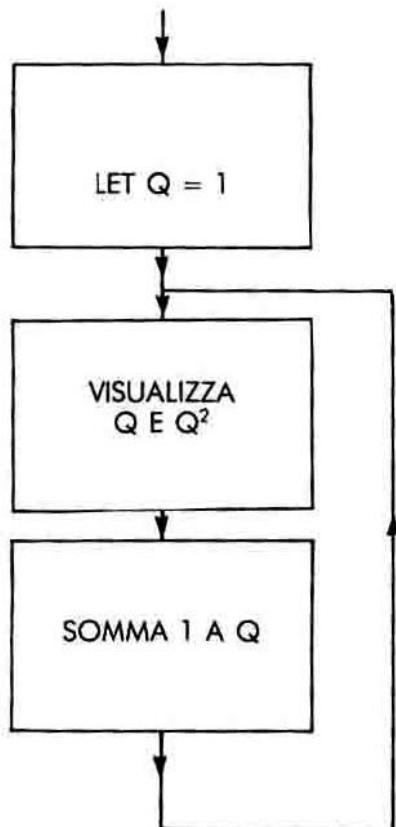
Un computer che esegue un programma è più o meno come un gioco dell'oca. All'inizio il giocatore va sul primo blocco. Ogniqualvolta l'azione descritta in un blocco è stata completata, il giocatore si sposta, lungo la riga con freccia, al blocco successivo.
Quando incontra un rombo, il giocatore esamina la condizione e decide se è vera. Se lo è, sposta la sua pedina al termine della linea VERO altrimenti segue la linea FALSO. Al limite raggiunge un blocco STOP che rappresenta la fine della partita.
Lo scopo di questa illustrazione è di aiutare a vedere due cose importanti sui computer:

- Un computer può eseguire soltanto una cosa alla volta (non parecchie).

- L'ordine in cui il computer esegue le cose è determinato dal programma.

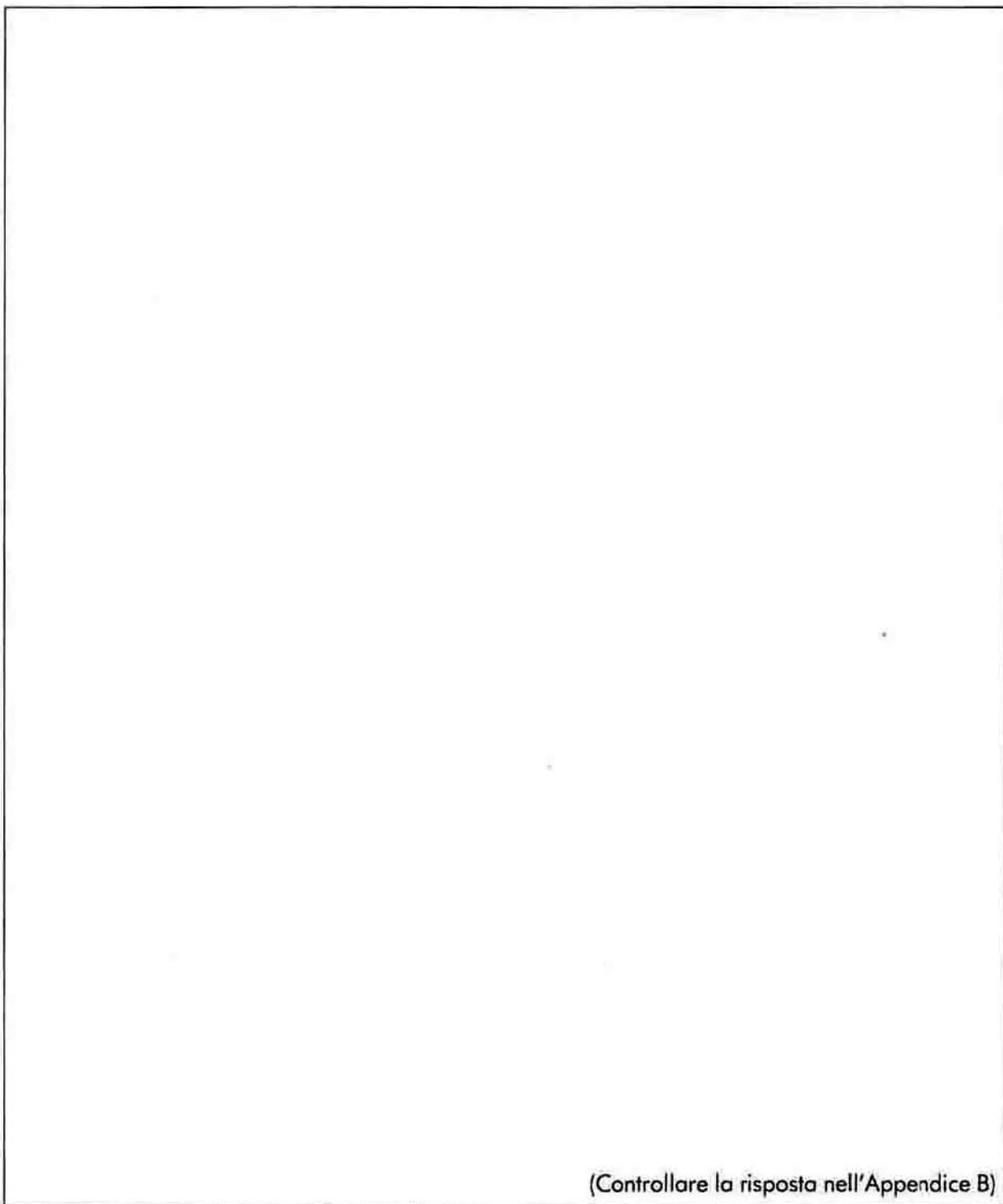
Spesso sorprende che non ci sia un simbolo per lo schema di flusso per un semplice comando GOTO. Ciò in quanto il GOTO non specifica qualsiasi azione: esso interessa soltanto l'ordine in cui i comandi sono eseguiti. Esso è ben rappresentato da una linea di collegamento. Per esempio:

```
10 Q=1

20 PRINT Q; Q*Q

30 Q=Q+1

40 GOTO 20
```

ha lo schema di flusso

Ora disegnare uno schema di flusso per il seguente programma. Usare un normografo di plastica per disegnare i blocchi:

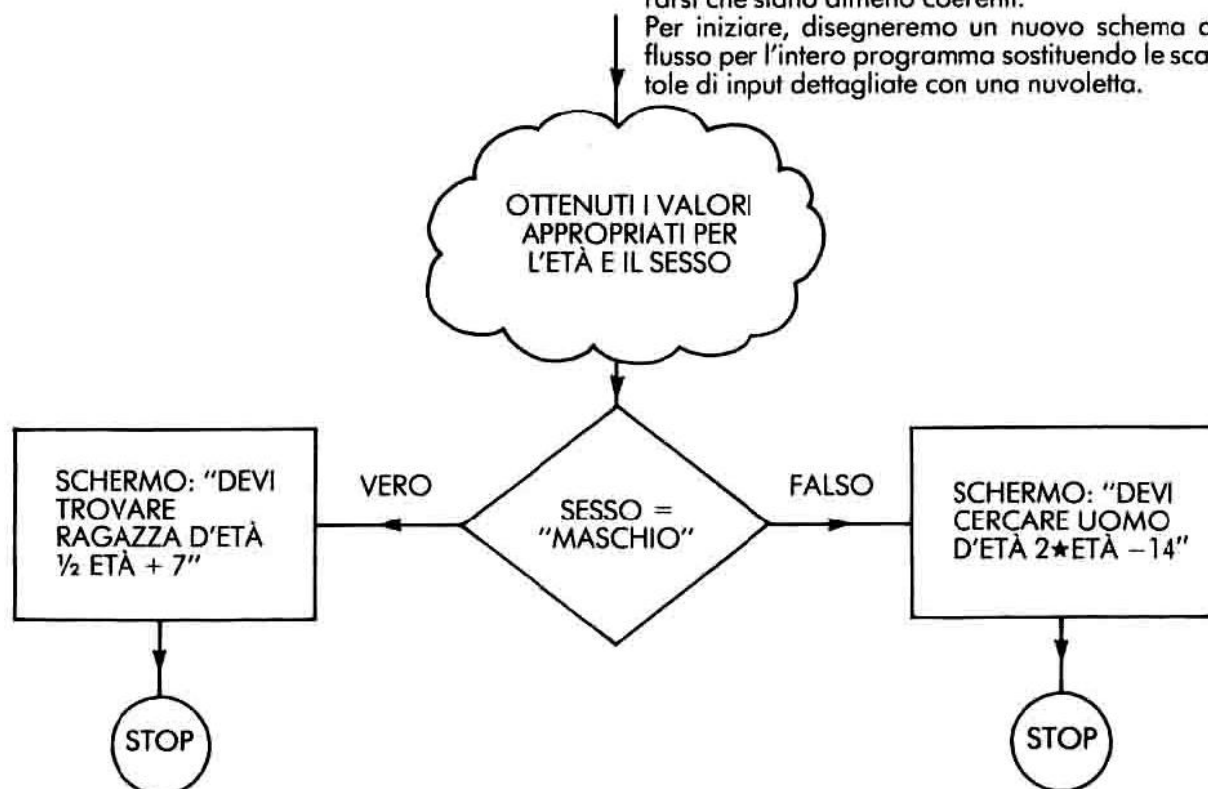
```
10 S=1
20 PRINT S,12*S
30 S=S+1
40 IF S<13 THEN 20
50 STOP
```

(Controllare la risposta nell'Appendice B)

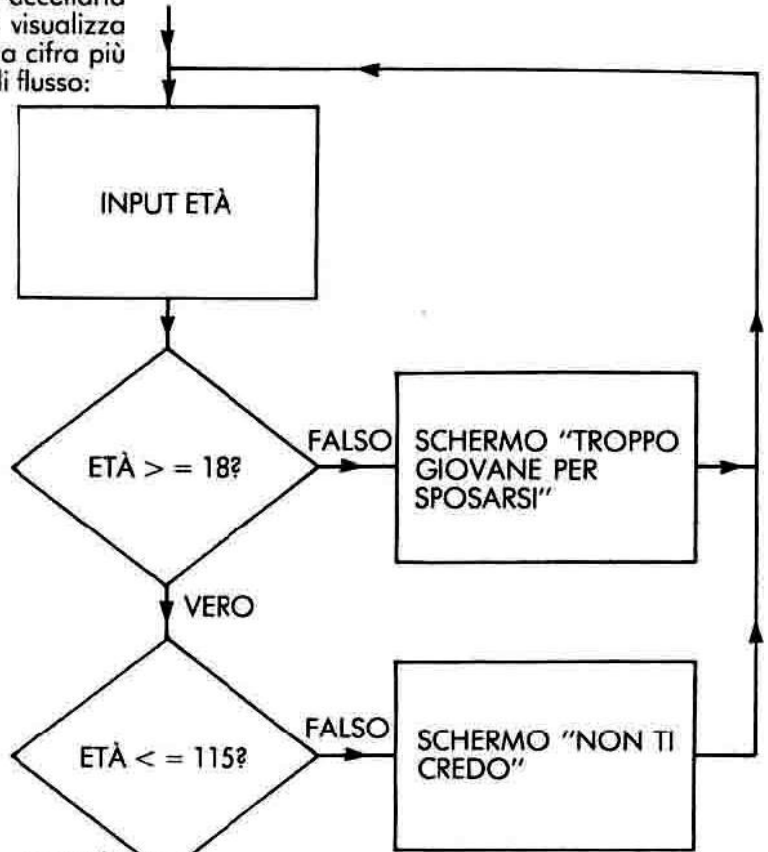
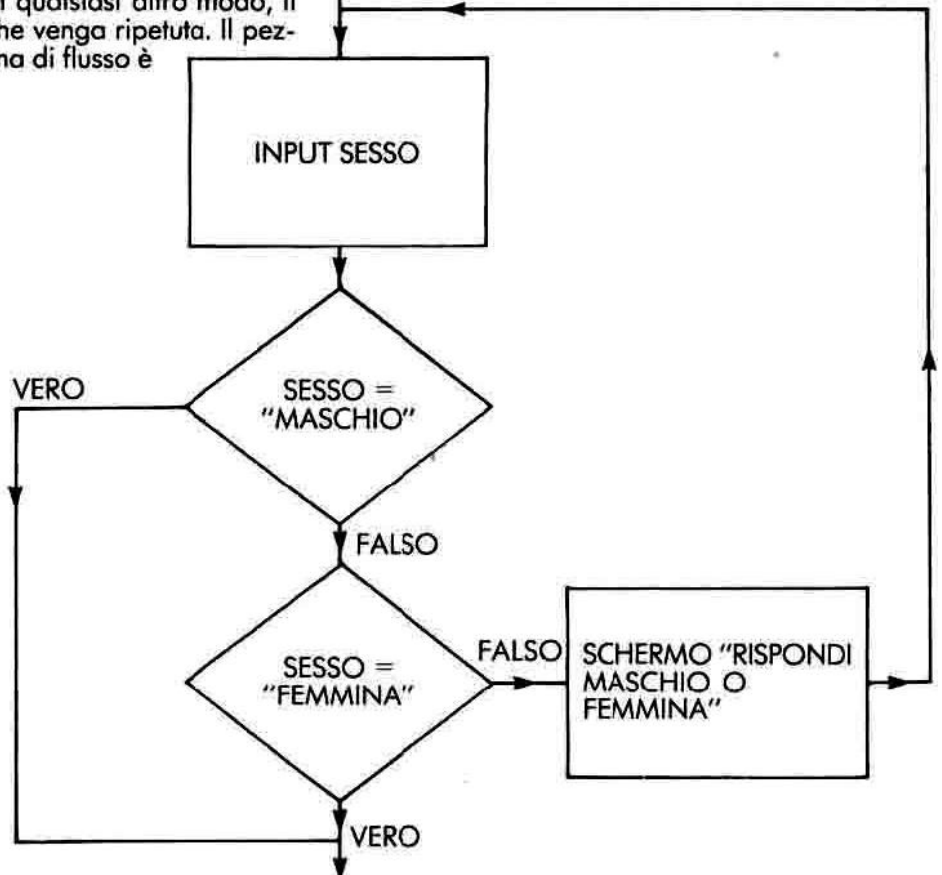
Esperimento 11.1 completato	

ESPERIMENTO 11·2

Faremo ora qualche ulteriore esplorazione. Una caratteristica del nostro programma per la guida al matrimonio è che se gli si forniscono dati scorretti dà risposte sciocche. Il nome per questo fatto è "GIGO" che sta "Garbage In, Garbage Out" (in altre parole, detto in italiano, "se si immette della porcheria si riceverà una porcheria"). Per esempio, una ragazza che indicasse la sua età in 6 anni si sentirebbe rispondere di trovarsi un marito con età di −2 anni: nemmeno un bagliore negli occhi dei suoi genitori! Inoltre, se l'utente dà qualsiasi risposta diversa da MASCHIO (maschio) nella seconda domanda, il programma presume che sia femmina. Qualcuno che risponde "M" o "UOMO" o "MASCHILE" o "RAGAZZO" si sentirebbe dire di trovarsi un uomo come partner.

C'è abbondanza di programmi che si comportano in questo modo stupido e che talvolta hanno dato al calcolo una reputazione cattiva del tutto immeritata. In pratica, è possibile evitare l'insorgere di questi problemi facendo passare le informazioni dell'utente attraverso un filtro per assicurarsi che siano almeno coerenti.

Per iniziare, disegneremo un nuovo schema di flusso per l'intero programma sostituendo le scatole di input dettagliate con una nuvoletta.

Usiamo una nuvoletta in quanto non abbiamo ancora fissato i dettagli di ciò che significa effettivamente "coerente". Usiamo la nuvoletta in quanto ci consente di pianificare il programma come un'intera unità, ma comporta l'elaborazione dell'azione in maggior dettaglio. Al punto in cui siamo ora arrivati la pianificazione è completa, ma ciò non significa che lo schema di flusso sia inutile o sbagliato.

Cosa significa dunque "coerente"? Per prima cosa pensiamo all'età dell'utente. Il valore inferiore probabile è 18 in quanto le persone al di sotto dei 18 anni non si recano spesso all'agenzia matrimoniale. Il limite superiore è difficile da stabilire, ma secondo il Guinness dei primati la più vecchia persona vivente ha 115 anni. Prenderemo questa cifra come guida.

Disegneremo un programma in modo che conoscendo l'età del cliente, decida se accettarla come ragionevole. In caso contrario visualizza una frase e invita il cliente a dare una cifra più realistica. Osservare questo schema di flusso:

Quando arriva il momento della seconda domanda, ci sono diversi modi con il quale il cliente potrebbe indicare se è maschio o femmina. In effetti, ce ne sono così tanti che non è possibile elencarli tutti e per contro fare in modo che il programma "riconosca" soltanto due parole MALE (maschio) e FEMALE (femmina). Se la risposta è data in qualsiasi altro modo, il programma chiederà che venga ripetuta. Il pezzettino corretto di schema di flusso è

Ora possiamo riunire questi due frammenti per avere uno schema di flusso completo per la nuvoletta che deve ricevere valori coerenti per AG e SX$.

INPUT ETÀ

ETÀ > = 18?

FALSO

SCHERMO "TROPPO GIOVANE PER SPOSARSI"

VERO

ETÀ < = 115?

FALSO

SCHERMO "NON TI CREDO"

INPUT SESSO

VERO

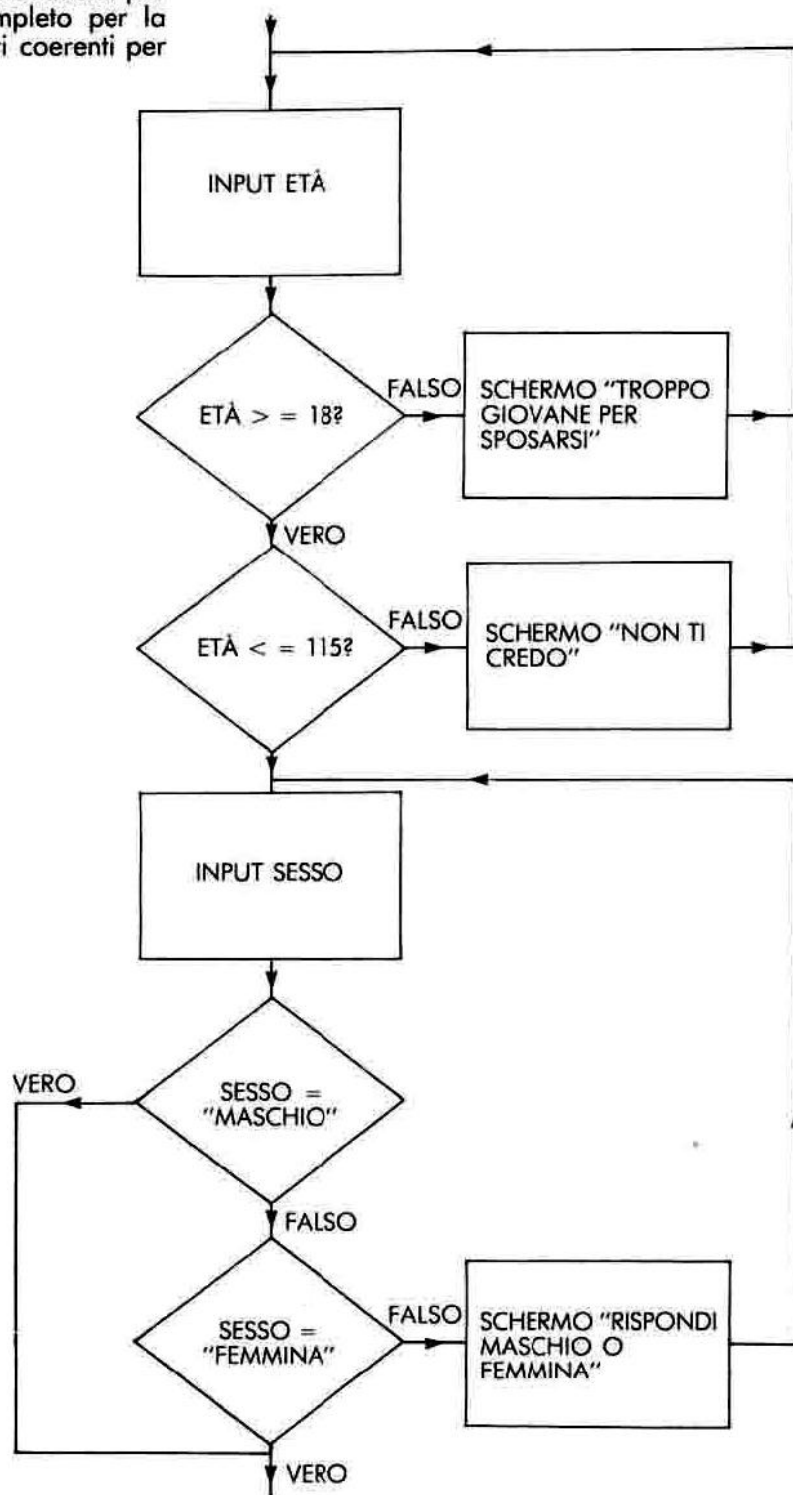
SESSO = "MASCHIO"

FALSO

SESSO = "FEMMINA"

FALSO

SCHERMO "RISPONDI MASCHIO O FEMMINA"

VERO

Una volta disegnata una serie di schemi di flusso, tradurli in programma è una cosa molto semplice. Inizieremo con lo schema di flusso principale, ma come primo blocco c'è una nuvola, così dobbiamo fare riferimento allo schema di flusso sussidiario e tradurlo. Torniamo indietro quindi allo schema principale per il resto del programma.
Si noterà che due dei rombi nello schema sussidiario hanno una linea VERO che va diritta al termine. Il modo più semplice per inserire i numeri di label dei corrispondenti comandi IF consiste nell'introdurre un REM (nota o commento) al termine della nuvoletta e usarlo come numero di label.

La REM non fa nulla, ma agisce come comodo punto di ancoraggio:

Si ottiene:

```
10 INPUT "QUANTI ANNI HAI"; AG
20 IF AG > = 18 THEN 50
30 PRINT "TROPPO GIOVANE PER SPOSARSI"
40 GOTO 10
50 IF AG < = 115 THEN 80
60 PRINT "NON TI CREDO"
70 GOTO 10
80 INPUT "MASCHIO O FEMMINA"; SX$
90 IF SX$="MASCHIO" THEN 130
100 IF SX$="FEMMINA" THEN 130
110 PRINT "RISPONDI MASCHIO O FEMMINA"
120 GOTO 80
130 REM AG E SX$ HANNO VALORI ADATTI
```

Che è seguito dal resto del programma come prima (ma con i numeri di label aggiustati).

```
140 IF SX$="MASCHIO" THEN 180
150 PRINT "DEVI CERCARE"
160 PRINT "UN UOMO DI"; 2*AG-14; "ANNI"
170 STOP
180 PRINT "DEVI TROVARE"
190 PRINT "UNA RAGAZZA DI"; AG/2+7;
    "ANNI"
200 STOP
```

Immettere questo programma nel **64** e provarlo. Per i valori coerenti dell'età, si comporterà esattamente come nella prima versione, ma si comporterà molto meglio nell'individuare e rifiutare risposte schiocche. Ha la qualità importante di robustezza, ossia la capacità di resistere agli abusi.
Per terminare questa unità, si scriverà un proprio programma. Prima di iniziare, ecco alcuni punti consigliati:

1. Munirsi di abbondante carta, di una matita e di una gomma. Spegnere il computer.
2. Studiare il problema accuratamente ed elaborare uno o due semplici esempi. Conservare la risposta per controllarla a fronte di quella del computer.
3. Iniziare decidendo quali variabili occorrono. Indicarne i nomi, i tipi e gli scopi in un "glossario". Per esempio, le variabili per il programma di consulenza matrimoniale verrebbero annotate come segue:

Nome	*Tipo*	*Scopo*
AG	Numero	Età del cliente
SX$	Stringa	Sesso del cliente ossia "MASCHIO" o "FEMMINA"

4. Disegnare uno schema di flusso per il programma. Bisogna essere preparati a fare numerosi errori e non sorprendersi se occorre ridisegnare lo schema quattro, cinque o sei volte. Continuare fino a che non si è soddisfatti. La programmazione è un lavoro duro e questa parte del lavoro — l'esecuzione dello schema di flusso — è dove comincia la maggior parte dello sforzo.
5. Tradurre ora lo schema di flusso in BASIC. Dovrebbe essere facile. Se non lo è significa che non si è disegnato lo schema di flusso correttamente per cui occorre tornare indietro e ricominciare da capo.
6. Ora — finalmente — accendere il **64** e immettere il programma. A parte qualche errore di battitura, dovrebbe girare senza preoccupazioni. Provarlo con numerosi esempi a piacere includendo uno di quelli precedentemente elaborati a mano. Infine, conservare il programma sul nastro (se lo si desidera) e conservare lo schema di flusso e il glossario delle variabili.

È stato appena ora descritto il modo in cui un buon professionista si prepara alla programmazione. Numerose persone non lo fanno per nulla — esse si siedono davanti ai computer e compongono i programmi direttamente sulla tastiera. Questo metodo talvolta funziona per piccolissimi problemi, ma solitamente porta a lunghi e incomprensibili programmi che funzionano soltanto qualche volta e che il programmatore trova impossibile modificare o riparare.
Occorre inoltre molto più tempo per far sì che tutto giri correttamente. In ogni caso, questo fatto non è del tutto ovvio — sembra infatti più rapido ignorare tutta la pianificazione e passare direttamente alla fase esecutiva. Questo in effetti è il motivo per cui tante persone programmano in maniera scorretta.
C'è una scelta: è possibile fare come suggerito e diventare rapidamente programmatori competenti oppure è possibile imparare nel modo più duro che ovviamente richiederà più tempo.

Ora eseguire la pianificazione, lo schema di flusso, la compilazione e la prova di un programma per il seguente problema:

In Ruritania, la tassa sulla casa è strutturata come segue:

Per ciascuna porta:	Lire 57
Per ciascuna finestra:	Lire 12
Per ciascun tetto coperto di paglia:	Lire 38
Per ciascun tetto ricoperto con tegole:	Lire 94

Supponendo che tutte le case devono essere o ricoperte con paglia o ricoperte con tegole, scrivere un programma per chiedere i particolari di qualsiasi casa e visualizzare la tassa da pagare. Per esempio, la risposta esatta per un cottage con tetto di tegole, con una porta e due finestre sarebbe di Lire (38+57+2★12) = Lire 119.
Fare in modo che il programma visualizzi gli importi delle tasse per le seguenti case (supponendo che tutte le porte e le finestre si trovino nella parte anteriore):

Controllare la risposta nell'Appendice B.

Esperimento 11.2 completato	

ESPERIMENTO 11·3

Caricare ed eseguire il programma UNIT11PROGR.
Dopo averlo listato, esaminare il codice e disegnare lo schema di flusso e il glossario relativi.

Esperimento 11.3 completato	

UNITA':12

Non occorre cercare molto a lungo in un programma per trovarvi un'iterazione in qualche punto. Le iterazioni sono così importanti che il linguaggio BASIC fornisce un metodo abbreviato per scrivere i dettagli essenziali.
Si ricorderà che ci sono quattro parti vitali per il controllo di qualsiasi iterazione:

- La scelta della variabile di controllo
- Il valore iniziale per la variabile di controllo
- Il valore finale per la variabile di controllo
- L'incremento o quantità di cui la variabile cresce ogni volta nel corso dell'iterazione.

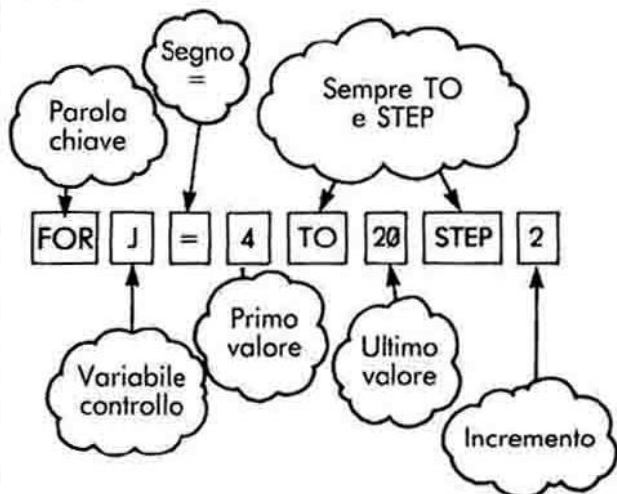
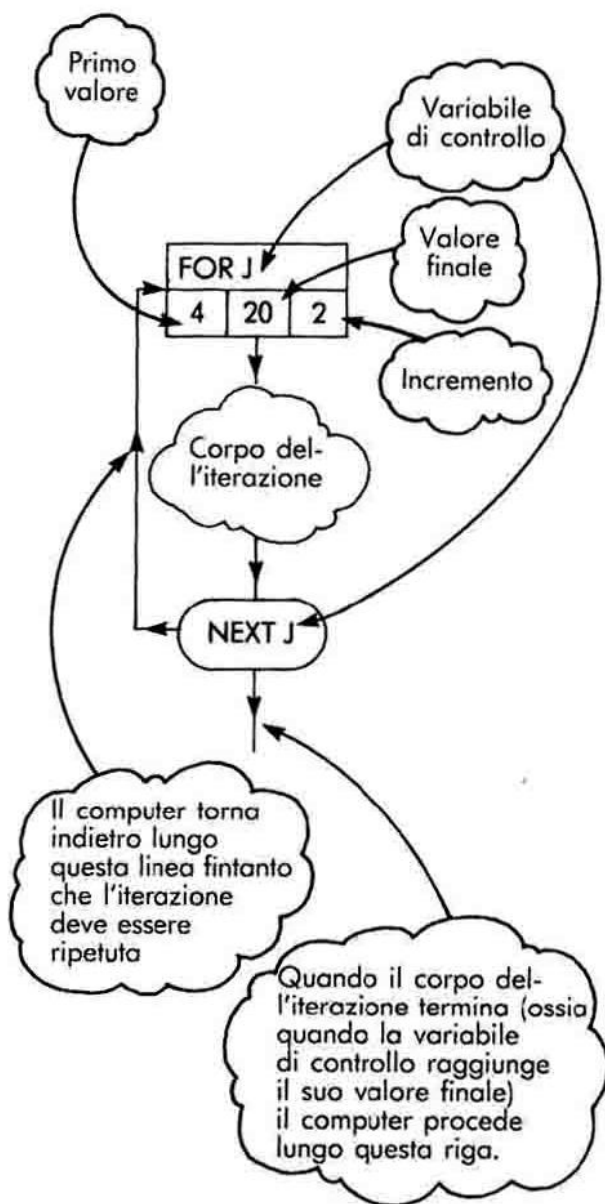
Tutte queste parti possono essere riunite in uno speciale comando che usa la parola chiave FOR, che è tutto ciò che occorre per impostare un'iterazione oltre beninteso al comando NEXT per contrassegnare la fine del corpo dell'iterazione.
Si confrontino i seguenti due programmi, che danno esattamente lo stesso risultato:

```
10 J=4                10 FOR J=4 TO 20 STEP 2
20 PRINT J,J★T        20 PRINT J,J★7
30 J=J+2              30 NEXT J
40 IF J<22 THEN 20
```

(Usando IF . . . . THEN) (Usando FOR . . . . NEXT)

In entrambi i casi

La variabile di controllo è	J
Il primo valore è	4
L'ultimo valore è	20
L'incremento è	2

L'esempio mostra come è costruito il comando FOR

Il comando NEXT cita il nome della variabile di controllo come riscontro per aiutare a leggere il programma.
Ogni comando FOR deve avere un corrispondente NEXT e tra di essi viene racchiuso il corpo dell'iterazione.
Negli schemi di flusso mostriamo le iterazioni in un modo speciale usando blocchi che non possono essere facilmente confusi con altri tipi di azione:

ESPERIMENTO 12·1

Per facilitare l'individuazione dei dettagli del comando FOR, osservare i seguenti programmi e scrivere ciò che si pensa che essi facciano comparire sullo schermo del **64.** Controllare quindi la risposta sulla macchina:

(i)
```
10 FOR Q=1 TO 16 STEP 5
20 PRINT Q;
30 NEXT Q
40 STOP
```

Previsione:

(ii)
```
10 FOR R=38 TO 50 STEP 3
20 PRINT R; 50-R
30 NEXT R
40 STOP
```

Previsione:

Tradurre ora il seguente programma in una notazione FOR-NEXT. Controllare la risposta eseguendo entrambe le versioni del **64** e assicurandosi che diano le stesse risposte:

```
10 PRINT "TABELLINA DEL NOVE"
20 S=1
30 PRINT S; "PER 9 =";
40 PRINT 9*S
50 S=S+1
60 IF S<13 THEN 30
70 STOP
```

Traduzione:

Ci sono alcuni punti che occorre ricordare a proposito dei comandi FOR e NEXT:

(a) Se l'incremento o la dimensione del passo è 1, lo "STEP 1" al termine del comando FOR può essere tralasciato. Il **64** comprende ugualmente cosa si vuole.

(b) Il controllo dell'iterazione può essere eseguito in modo da contare alla rovescia usando un valore di incremento negativo. Il programma

```
10 FOR X=10 TO 5 STEP -1
20 PRINT X;
30 NEXT X
```

visualizzerà: `10 9 8 7 6 5`

nell'ordine.

(c) Il corpo dell'iterazione è sempre eseguito almeno una volta anche se il valore finale è minore del valore iniziale. Per esempio:

```
10 FOR R=5 TO 3
20 PRINT R
30 NEXT R
```

visualizzerà `5`

(d) I valori nel comando FOR non devono essere necessariamente numeri ma possono essere espressione che comprendono altre variabili.

Per esempio, il seguente programma visualizzerà il numero dei simboli "cuore" richiesti dall'utente. Provarlo e studiarlo accuratamente:

```
10 INPUT "QUANTI CUORI"; H
20 FOR K=1 TO H
30 PRINT " [CTRL] e [RED] ♥";
40 NEXT K
50 STOP
```

(e) La variabile di controllo non può essere una stringa. Per esempio, il "comando"

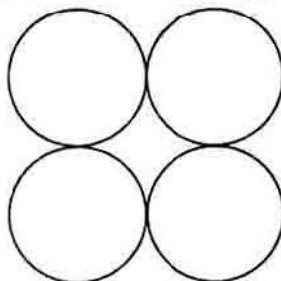
darebbe un errore di sintassi e non è pertanto ammesso usare questa costruzione. Conoscendo ciò, prevedere il risultato dei seguenti programmi e controllare i risultati sul computer:

(i)
```
10 FOR A=1 TO 4
20 PRINT A*A
30 NEXT A
40 STOP
```

(ii)
```
10 FOR B=3 TO 0 STEP -1
20 PRINT B;
30 NEXT B
40 STOP
```

(iii)
```
10 FOR C =5 TO 4
20 PRINT C;
30 NEXT C
40 STOP
```

(iv)
```
10 X=5
20 Y=9
30 Z=2
40 FOR W=X TO Y STEP Z
50 PRINT W;
60 NEXT W
70 STOP
```

Finora ci si è concentrati sui dettagli dei comandi FOR e NEXT, scegliendo accuratamente i corpi delle iterazioni che vengono controllate in modo da farli risultare i più semplici possibili. In pratica il corpo di un'iterazione deve essere necessariamente corto e semplice, ma può essere complesso a piacere — ricordando però che viene eseguito ogni volta che il computer esegue l'iterazione stessa. Si supponga che venga chiesto di costruire una piramide a base quadrata costituita da palle di cannone. Si numereranno gli strati 1, 2, 3... iniziando dalla cima. Lo strato 1 sarà la punta, per la quale occorrerà soltanto una palla di cannone. Lo strato 2 sarà il secondo e occorreranno 4 palle disposte come segue:

Lo strato 3 richiederà 9 palle, lo strato 4 — 16 e così via.
Chiaramente il numero di palle da cannone che occorre per l'intera piramide dipende da quanti strati si intendono costruire. Una piramide a tre strati ha bisogno di 1+4+9 ossia 14 palle da cannone; una con 4 strati ne richiederà 1+4+9+16 ossia 39.
Se si ha in programma di costruire una piramide molto ampia, queste somme risulteranno piuttosto lunghe e noiose e si potrebbe pertanto utilizzare un programma di computer per eseguirle. Questo programma risponde alla domanda "quante palle da cannone occorrono per una piramide con 'N' strati?".
Nel disegnare il programma, un fattore chiave è il numero di palle da cannone per ciascun strato. I numeri 1, 4, 9 e 16... sono abbastanza familiari e in effetti si scopre che il numero delle palle da cannone di ciascun strato è il quadrato del numero che contraddistingue lo strato. Per esempio, lo strato 7 avrà bisogno di 7×7 ossia 49 palle da cannone.
Ora i dettagli per il programma. Si inizierà pensando alle variabili necessarie.
Il piano globale prenderà in considerazione gli strati un per uno. Si farà cioè in modo che il computer calcoli quante palle occorrono per ogni strato e si sommerà questo numero ad un "totale mobile". All'inizio, il totale mobile deve essere posto a zero. Al termine, quando sono stati presi in considerazione tutti gli strati, il totale mobile indicherà il numero di palle da cannone desiderate per l'intera piramide. Questa è la risposta al problema.

Un nome adatto per il totale mobile è RT.

Occorrono due altre variabili:

(i) Il numero di strati nella piramide. Ricordarsi che il programmatore non conosce questo numero; tocca all'utente fornire qualsiasi valore desiderato. Un buon nome per questa variabile è L.

(ii) Quando il programma viene eseguito affronterà prima lo strato 1 quindi lo strato 2, lo strato 3 e così via. Occorre una variabile per

per indicare con quale dei diversi strati L il programma sta operando in qualsiasi momento. Un nome adatto di variabile è V. Dato che V prenderà tutti i valori compresi tra 1 e L, il numero dello strato inferiore, si può indovinare che sarà la variabile di controllo in un comando FOR, pertanto:

```
FOR V = 1 TO L
.......
NEXT V
```

Il glossario per il nostro programma è di conseguenza:

Nome	*Scopo*
RT	Per conservare il totale mobile delle palle da cannone
L	Numero degli strati nella piramide
V	Numero dello strato col quale il programma opera in qualsiasi momento

Successivamente si scriveranno alcune delle azioni che il programma deve intraprendere:

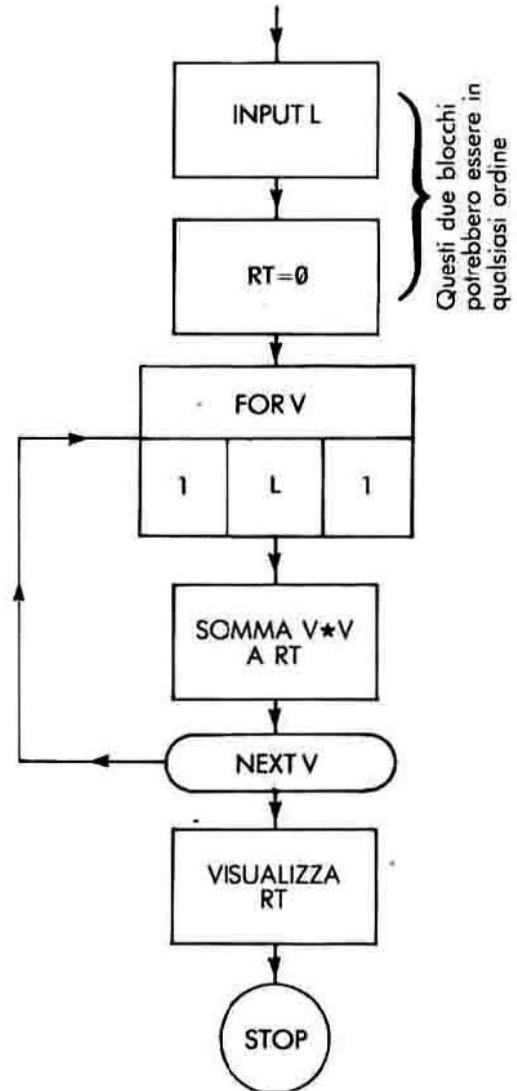
Aggiungere V elevato al quadrato a RT	(Ciò somma il numero delle palle da cannone per lo strato numero V)
Stampare RT	(Visualizza i risultati)
Definire RT=Ø	(Azzera RT)
Immettere L	(Chiede all'utente quanti strati ci sono nella sua piramide)
FOR V=1 TO L NEXT V	(Iterazione di controllo per tenere conto di ogni strato)
STOP	

Questi sono i frammenti del programma occorrenti, che devono però essere messi insieme nell'ordine giusto. Si è già deciso che deve esserci un'iterazione e sarà di grande aiuto per poter dire, per ciascun comando, se deve essere eseguito

prima che inizi l'iterazione

o all'interno dell'iterazione (come parte del corpo dell'iterazione)

o dopo che l'iterazione è stata terminata

Si possono usare vari segnali. Il programma deve sapere quante volte eseguire l'iterazione, prima che l'iterazione stessa possa iniziare, così l'input L deve venire *prima* dell'iterazione. Altrettanto deve fare il comando che pone RT a zero. Il numero totale di palle da cannone per l'intera piramide ne comprende sempre alcune per ciascun strato. Il comando per aggiungere il valore di uno strato a RT deve essere ripetuto molte volte per cui va all'interno dell'iterazione. Infine, il computer non può fornire la risposta esatta fino a che non ha preso in considerazione tutti gli strati, per cui il comando PRINT può venire usato solo dopo che l'iterazione è terminata. Siamo ora arrivati a un punto dove è possibile disegnare uno schema di flusso. Esso è

E il corrispondente programma è

```
10 INPUT "NUMERO STRATI"; L
20 RT=Ø
30 FOR V=1 TO L
40 RT=RT+V★V
50 NEXT V
60 PRINT RT; "PALLE CANNONE RICHIESTE"
70 STOP
```

Immettere questo programma e provarlo.
Ora un altro problema. Nel gioco del cricket, un giocatore può avere un numero di "innings" o

turni durante la stagione. Ogni volta egli segna qualche "run" o corsa: molte se è un giocatore valido o fortunato, poche (o addirittura nessuna) se non è nè valido nè fortunato. Se si vuole conoscere come un giocatore si è comportato per l'intera stagione, è possibile calcolare il numero medio di "run" per ogni "inning". Ciò si ottiene sommando tutti i run acquisiti durante la stagione e dividendo il risultato per il numero degli innings. Per esempio, se il giocatore gioca tre volte e segna 20, 30 e 70, la sua media è (20+30+70):3 ossia 40 run per inning.
Si consideri un programma che esegue questo calcolo. Esso deve chiedere il numero degli innings e quindi i relativi punteggi per poterli sommare. La presentazione completa sarebbe la seguente:


```
RUN

NUMERO INNINGS? (3)

PUNTI? (20)

PUNTI? (30)

PUNTI? (70)

MEDIA =40
```

Numeri battuti dall'utente

Il compito stavolta consiste nello scrivere il programma per questo problema. Per renderlo più facile, si forniranno il glossario e tutti i comandi, ma in ordine alterato e privi di label. Iniziare diegnando uno "scheletro" con i comandi delle iterazioni quindi inserire gli altri comandi nei posti esatti. Infine, eseguire il programma sul **64** e assicurarsi che funzioni. Se si blocca, osservare la risposta corretta nell'Appendice B, ma ricordarsi che così facendo si ammette l'insuccesso! Ecco il glossario e i comandi rimescolati:

Nome	*Scopo*
J	Numero di innings durante la stagione
Q	Varibile di controllo per l'iterazione
RS	Usato per sommare i run totali segnati
S	Punteggio per ciascun inning

```
NEXT Q

INPUT"NUMERO INNINGS"; J

INPUT"PUNTI"; S

PRINT "MEDIA="; RS/J

STOP

FOR Q=1 TO J

RS=RS+S
```

Esperimento 12.1 completato	

ESPERIMENTO 12·2

Termineremo questa sezione con un problema che occorrerà risolvere senza aiuto. Se si va alla posta, è facile rimanere bloccati in una coda perchè qualcuno deve comprare una grande quantità di francobolli. Lo si sente dire all'impiegato dello sportello:

"ottantatre da 11 lire

e centodiciassette da 14 lire

e trentacinque da 75 lire"

e così via. Quando tutti i francobolli sono stati contati, all'impiegato serve un tempo notevole per calcolarne il costo totale.
Scrivere un programma per aiutare l'impiegato. La visualizzazione dovrebbe apparire più o meno come questa:

```
RUN

NUMERO BLOCCHI? (4)

BLOCCO 1

  NUMERO BOLLI? (20)

  VALORE UNITARIO? (15)

BLOCCO 2

  NUMERO BOLLI? (5)

  VALORE UNITARIO? (75)

BLOCCO 3

  NUMERO BOLLI? (4)

  VALORE UNITARIO? (1)

BLOCCO 4

  NUMERO BOLLI? (100)

  VALORE UNITARIO? (14)

TOTALE = 2079 LIRE
```

Numeri battuti dall'utente

Il programma dovrebbe comprendere 10 comandi (compreso lo STOP). Quattro di questi comandi formeranno il corpo di un'iterazione, eseguita una volta per ogni blocco. In ogni caso non tentare di scrivere il programma senza avere un appropriato disegno, glossario e schema di flusso. Prendersi tempo a sufficienza.
Se dopo aver speso buona quantità di sforzi non si è in grado di risolvere il problema correttamente, tornare indietro di qualche unità e cioè fino al punto in cui ci si sente sicuri e riprendere di lì lo studio del materiale del corso.
Confrontare infine la risposta con quella indicata nell'appendice B.

Esperimento 12.2 completato	

Il quiz di auto-test per l'Unità 12 è detto "UNIT12QUIZ".

UNITA':13

EXPERIMENT 13•1

Una delle caratteristiche più interessanti del COMMODORE 64 è la capacità di creare numerosi suoni musicali ed altri rumori interessanti. La macchina è munita di un sintetizzatore simile a quello dei moderni complessi, con generatori di suoni, filtri e funzioni speciali che si possono controllare attraverso un programma scritto in BASIC.
Il sintetizzatore è un dispositivo complesso ed anche un musicista esperto si troverebbe a disagio se volesse sfruttarlo in tutte le sue possibilità. Fortunatamente noi siamo in grado di fare molto con un semplice approccio ed è di questo che ci occuperemo in questa unità. Una spiegazione più completa del sintetizzatore figurerà nella 2ª Parte di questa serie.
Per avere un'idea della gamma di effetti sonori che è possibile creare, caricare il programma intitolato "SOUND DEMO" ed eseguirlo. Assicurarsi che il volume del televisore sia opportunamente regolato, altrimenti si corre il rischio di non sentire nulla.
Se si gradisce qualcuno di questi suoni non è difficile usarli in un proprio programma. Al termine di questa Unità compare un listato completo di SOUND MEMO e basta estrarre il gruppo di comandi che corrispondono al suono scelto e copiarli nel proprio programma. Ciascun gruppo è racchiuso tra istruzioni REM del tipo:

```
2000 REM ★★★ HEART BEATING ★★★
       .....
       .....
       .....
2150 REM ★★★ END OF HEART BEATING★★★
```

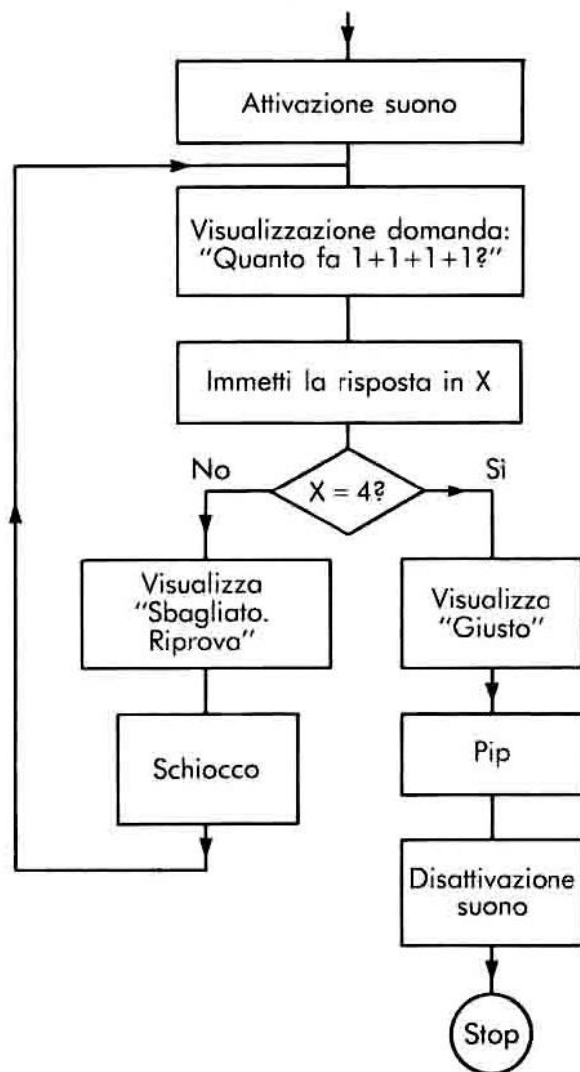
Comandi da copiare

Non copiare i comandi REM o il RETURN che segue il REM finale. Sarà possibile mantenere inalterati questi numeri di riga ma qualora li si docesse modificare, ricordarsi di occuparsi degli IF e dei GOTO, che devono essere variati di conseguenza. Non occorre comprendere il significato dei comandi copiati: basta assicurarsi di copiarli esattamente. Se si commette un errore (specialmente nel copiare numeri tipo 54272 o inserendo V invece di VV), si può danneggiare il programma di controllo, provocando un comportamento strano della macchina che non si elimina se non spegnendola e riaccendendola. Controllare quindi prima di eseguire.
Una raccomandazione finale: la macchina emetterà dei suoni solo se si sarà acceso in precedenza in generatore di suoni. Ciò avviene col comando

```
POKE 54295,0: POKE 54296,15
```

che dovrebbe essere incluso all'inizio di ogni programma che usa il sintetizzatore. Il programma dovrebbe sempre terminare disattivando il generatore di suoni. Il comando adatto è

```
POKE 54296,0
```

Ecco un semplice esempio. Il programma è strutturato per porre all'utente una domanda. Se viene data la risposta esatta, viene emesso un "pip" di gradimento e si ottiene un messaggio di approvazione. Se la risposta è sbagliata, la macchina emette uno schiocco invitando a riprovare. Lo schema di flusso è il seguente:

e il codice corrispondente è:

```
   10 POKE 54295,0 : POKE 54296,15
```

(attivazione suono)

```
   20 PRINT "QUANTO FA 1+1+1+1";
   30 INPUT X
   40 IF X 4 THEN 3 8
   50 PRINT "GIUSTO"

3010 VV = 54272
3020 POKE VV+6,8 : POKE VV+5,31
3030 POKE VV+1,18  : POKE VV+4,33
3040 FOR MM = 1 TO 100 : NEXT MM
3050 POKE VV+4,0
```

(Comandi per creare un PIP, copiati dal listato di SOUND MEMO)

```
3060 POKE 54296,0 (Disattivazione suono)

3070 STOP
```

(Fine del programma)

```
3080 PRINT"SBAGLIATO. RIPROVA"
4010 VV = 54272 : POKE VV+6,240
4020 POKE VV+1,4 : POKE VV+5,0 : POKE
     VV+4,33
4030 FOR NN = 1024 TO 512 STEP-8
4040 POKE VV+1,NN/256 : POKE VV,NN AND
     255
4050 NEXT NN
4060 POKE VV+4,0
```

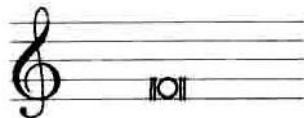
(Comandi per creare uno schiocco copiati dal listato di SOUND MEMO)

```
4070 GOTO 20
```

Ora esamineremo come vengono effettivamente generati i suoni. La spiegazione sarà un poco tecnica e se si hanno dei problemi di interpretazione la si può saltare tranquillamente e passare all'unità successiva.
Iniziamo con le note musicali. Ogni nota ha *cinque* qualità o *attributi*, che la fanno suonare in modo diverso dalle altre:

(a) *L'altezza.* Ci sono note alte e note basse. Nella notazione musicale l'altezza è solitamente indicata dal nome della nota (Si o Do) o dalla sua posizione sul pentagramma:

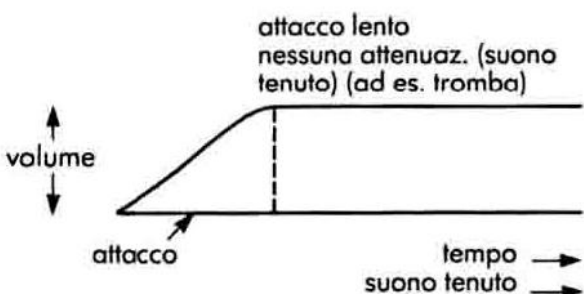
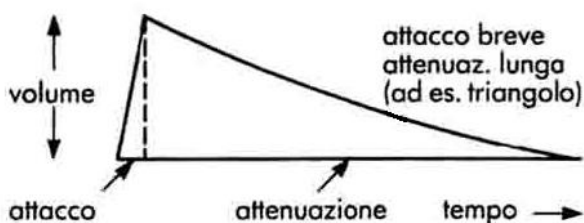
L'altezza di qualsiasi nota è direttamente correlata alla sua *frequenza*, che è il numero di vibrazioni che arrivano all'orecchio ogni secondo. Il Do centrale ha una frequenza di circa 256 vibrazioni al secondo e la sua frequenza si raddoppia per ogni ottava di cui si sale (e si dimezza per ogni ottava di cui si scende).

(b) La *durata*. Alcune note suonano più a lungo di altre. Nella notazione musicale, la lunghezza di una nota è solitamente indicata dal simbolo usato:

In pratica un simbolo musicale non indica sempre esattamente la stessa lunghezza, per cui si troverà più comodo misurare la durata di una nota in millisecondi (1/1000 di secondo).

(c) Alcune note musicali sono "tenute", vale a dire che proseguono con lo stesso grado di intensità finchè vengono suonate. Gli strumenti a fiato ed i membri della famiglia dei violini creano generalmente note tenute. Altri suoni, come quelli creati percuotendo o pizzicando sono detti transienti, in quanto iniziano intensamente ma si attenuano gradualmente fino a scomparire. Il pianoforte è un buon esempio di strumento con note transienti: una volta premuto un tasto il suono si attenua quantunque il tasto rimanga abbassato.

(d) Ogni nota ha un valore di "attacco", che descrive la rapidità con la quale raggiunge il massimo volume. Una corda pizzicata con un oggetto duro, ad esempio un plettro, raggiunge pressochè istantaneamente il massimo volume, come del resto fa uno strumento a percussione colpito da un martelletto metallio. Per contro, se la corda è pizzicata con le dita o compita da un martelletto morbido, ha bisogno di più tempo. Un tempo ancora maggiore è richiesto da un strumento a fiato per dare inizio alla vibrazione dell'aria. Naturalmente anche l'attacco più lungo dura solo una piccola frazione di secondo, per cui non ce se ne accorge quando la nota inizia. In compenso influisce notevolmente sulla qualità del suono che viene udito.

(e) Le note transienti hanno anche una velocità di "attenuazione", che non è altro che la misura del tempo che il volume impiega a scomparire una volta che la nota è stata suonata. Il tempo di attenuazione è molto superiore a quello di attacco, ed è solitamente misurato in secondi. Una breve attenuazione rende ovattato il suono di una nota mentre una lunga lo rende squillante.

Le informazioni relative al livello di tenuta ed alla velocità di attacco e di attenuazione sono spesso presentate sun un grafico detto "inviluppo". Si tratta di un termine tecnico che descrive il modo in cui l'intensità di una nota varia nel tempo durante il quale viene eseguita. Qui di seguito sono presentati tre diversi inviluppi tipici:

(e) Infine ogni nota ha un timbro o qualità tonale. Può essere morbido come quello di un'arpa o stridulo come quello di una tromba.

Nel corso di una pagina abbiamo così citato otto termini tecnici: altezza, frequenza, durata, livello di tenuta, attacco, attenuazione, inviluppo e timbro. Nessuna meraviglia quindi se fino a questo punto la materia sembra un poco dificile. Il programma ESG (Experimental Sound Generator) è studiato per aiutare a chiarire questi concetti ed a distinguere il modo in cui le variazioni di frequenza, velocità di attenuazione, ecc. influiscono in realtà suoi suoni prodotti. Caricare il programma e battere la barra di spazio a piacere. La nota emessa ha gli attributi visualizzati sullo schermo:

Frequenza: 512 vibrazioni al secondo
Durata: 250 millisecondi
Livello di tenuta: 0
Valore di attacco: (istantaneo)
Valore di attenuazione: 9 (circa 1/4s)
Tipo di forma d'onda: triangolare (che è il nome di un tipo di timbro piuttosto morbido).

È ora possibile variare gli attributi, uno alla volta, e rendersi conto di ciò che succede. Cambiare dapprima la frequenza battendo F quindi il numero 1024 seguito da un RETURN
Si udrà a questo punto l'altezza salire di un'ottava (poichè 1024 è esattamente il doppio di 512). Cambiare la frequenza, ad esempio in 700. Ciò provoca un abbassamento dell'altezza. È possibile variare l'altezza tra 20 e 4000 vibrazioni vibrazioni al secondo.

Provare ora a cambiare la durata. Il programma accetta qualsiasi valore compreso tra 1 e 5000 millisecondi. Se la durata è inferiore al tempo di attenuazione, l'inviluppo viene ridotto e non si riesce ad udire l'intera qualità sonora della nota. Per rendersi conto di questo effetto occorre ridurre la durata della nota a 50 millisecondi.
Il "livello di tenuta" ha soltanto due valori realmente interessanti: 0 per una nota con attenuazioni e 15 per una nota tenuta. Provare gli effetti di entrambe le regolazioni.
Il valore di attacco opera in maniera diversa da quella che ci si potrebbe aspettare: maggiore è il numero e più lento è l'attacco. Valori di attacco da 0 a 2 riproducono gli effetti dei veri strumenti musicali, mentre valori più elevati sono caratterizzati dalla tipica sensazione della musica elettronica. Provare 'A=6' ad esempio!
Provare ora vari valori di attenuazione compresi tra 6 (suono ovattato) e 11. Per i tempi di attenuazione più lunghi, aumentare la durata della nota in modo da udire l'intero inviluppo.

Infine varrebbe la pena di esplorare i quattro timbri o "forme d'onda" come sono spesso chiamati, che sono:

- "Triangolare" (che si ottiene battendo "T")
  — suono stridulo
- "A impulsi" (battere "S")
  — un suono rauco che si può ulteriormente variare agendo sull'ampiezza dell'impulso tra 8 e 999. Le informazioni sull'ampiezza dell'impulso appaiono non appena si sceglie la forma d'onda corrispondente.
- "Rumore" (battere "N")
  — questa forma d'onda non produce una nota musicale ma un rumore impetuoso, che si può usare per esplosioni, rumore di *aviogetti*, respiro affannoso ed effetti simili. Le altre regolazioni funzionano sempre, per cui è possibile personalizzare il rumore a secondo delle esigenze.

È opportuno dedicare un poco di tempo a variare i valori degli attributi ed a verificare i risultati, in modo da avere un'idea dei vari effetti. Provare quindi a creare dei propri suoni usando la tabella che segue (nella quale sono già state compilate alcune righe a titolo di esempio).

TIPO DI SUONO	FREQUENZA	DURATA	LIVELLO TENUTA	VALORE ATTACCO	VALORE ATTEN.	FORMA D'ONDA	AMPIEZ. IMPULSO
ARPA	500	1000	0	1	11	TRIANG.	—
SIRENA DI NAVE	80	2500	15	7	0	A IMPULSI	999
ESPLOSIONE	200	2000	0	0	12	RUMORE	—
PASSAGGIO JET	4000	5000	0	10	13	RUMORE	—
TROMBA	1000	500	15	3	0	A IMPULSI	20
OBOE							
PIZZICATO (stringa di violino pizzicata)							
FISCHIO POLIZIA							
TRIANGOLO							
TAMBURO							

Una volta disegnato un suono definendone gli attributi (frequenza, durata, ecc.) lo si desidera normalmente inserire nel programma che si sta scrivendo. Il sintetizzatore è controllato dalle istruzioni POKE, più o meno come i colori del riquadro e dello sfondo. Lo schema di flusso per creare un singolo suono è il seguente:

Le regole per convertire gli attributi in comandi sono molto lineari e si impareranno presto:

(a) La frequenza è controllata dagli indirizzi 54272 e 54273. Se la frequenza desiderata (in vibrazioni al secondo) è una variabile detta F, i comandi necessari sono:

```
FF = 16*F
POKE 54273,FF/256
POKE 54272, (FF-32768) AND 255
```

(b) L'ampiezza dell'impulso (se occorre) è impostata all'indirizzo 54275. Per un valore di ampiezza d'impulso di P, il comando adatto è.

```
POKE 54275,P/8
```

(c) Il livello di tenuta è impostato in 54278. Per suoni tenuti battere
POKE 54278,240
Per suoni che si attenuano battere
POKE 54278,0

(d) I livelli di attacco e di attenuazione sono entrambi controllati da 54277. Il numero che si inserisce è pari a 16 volte la velocità di attacco più la velocità di attenuazione. Per i valori A e D scegliere:
POKE 54277,16*A+D

Questi comandi completano la preparazione del suono e si possono scrivere in qualsiasi ordine. Il suono vero e proprio viene attivato inserendo (POKE) un numero speciale in 54276. Il numero controlla le forme d'onda. In particolare per le forme d'onda:

triangolare	POKE 54276,17
a dente di sega	POKE 54276,33
a impulsi	POKE 54276,65
di rumore	POKE 54276,129

Ora il programma attende la durata della nota, ad esempio T millisecondi, che viene impostata con un'iterazione FOR-NEXT che non fa altro che contare:

```
FOR C=1 TO T
NEXT C
```

Infine il suono deve essere interrotto inserendo (POKE) 0 all'indirizzo 54276:
POKE 54276,0

Ora si possono riunire tutti i dettagli in un esempio. Si supponga di voler scrivere un programma che in un certo punto imiti la sirena di una nave. I valori necessari sono indicati nella seconda riga della tabella di pagina 106.

```
500 FF=16*80
510 POKE54273,FF/256
520 POKE54272, (FF-32768) AND255
530 POKE54275, 999/8
540 POKE54278,240
550 POKE54277,16*7 + 0
560 POKE54276,65
570 FORC = 1 TO 2500
580 NEXTC
590 POKE54276,0
```


Ciò:
Imposta la frequenza (righe 500-520)
Imposta l'ampiezza dell'impulso (riga 530)
Imposta il livello di tenuta (riga 540)
Imposta i valori di attacco e di attenuazione (riga 550)
Inizia la nota (riga 560)
Attende 2500 millisecondi (riga 580)
Interrompe la nota (riga 590)

I numeri di riga (da 500 a 590) sono abbastanza arbitrari, in quanto potrebbe andar bene anche qualsiasi altra serie consecutiva. Ricordarsi in ogni caso di attivare il generatore di suoni all'inizio del programma.

Per terminare questa Unità verranno ora illustrati alcuni effetti sonori che non si possono in alcun senso considerare note musicali e che dipendono dalla variazione della frequenza del generatore mentre il suono viene effettivamente prodotto. L'effetto è indescrivibile in parola ma va semplicemente ascoltato.
Impostare ed eseguire il seguente programma:

```
10 POKE54296,15
20 POKE54278,240
30 POKE54277,0
40 POKE54276,33
50 FORN = (10) TO (80)
60 POKE54273 ,N
70 NEXTN
80 GOTO50
```

Quando si è ascoltato abbastanza, premere RUN STOP e RESTORE contemporaneamente per interrompere il programma.
È possibile alterare la qualità del suono cambiando i numeri racchiusi nel circoletto oppure usando 17 o 129 invece di 33 nel comando 40.

Si può ottenere un profilo sonoro ancor più complesso inserendo l'iterazione di controllo in un'altra iterazione in modo che i valori di inizio e di fine cambino nel tempo:

```
 10 POKE54296,15
 20 POKE54278,240
 30 POKE54277,0
 40 POKE54276,33

 50 FORC= 1 TO 20
 60 FORN =(40 - C)TO(25 - C)STEP - 1
 70 POKE 54273,N
 80 NEXTN
 90 NEXTC
100 GOTO50
```

Anche in questo caso è possibile variare i valori entro i circoletti oppure è possibile far procedere le iterazioni (una o l'altra o entrambe) in direzioni opposte.
si spera che questa Unità abbia destato l'interesse del lettore per la creazione di suoni col sintetizzatore del COMMODORE 64. Questo settore della programmazione è unico, in quanto non ci sono "risposte corrette". La sola cosa che conta è che valga la pena di ascoltare i suoni che vengono prodotti.
Per concludere l'Unità, provare a programmare una sirena o un clacson variando l'altezza del suono prodotto. Se la cosa riesce, conservare il programma per uso futuro.

SOUND DEMO

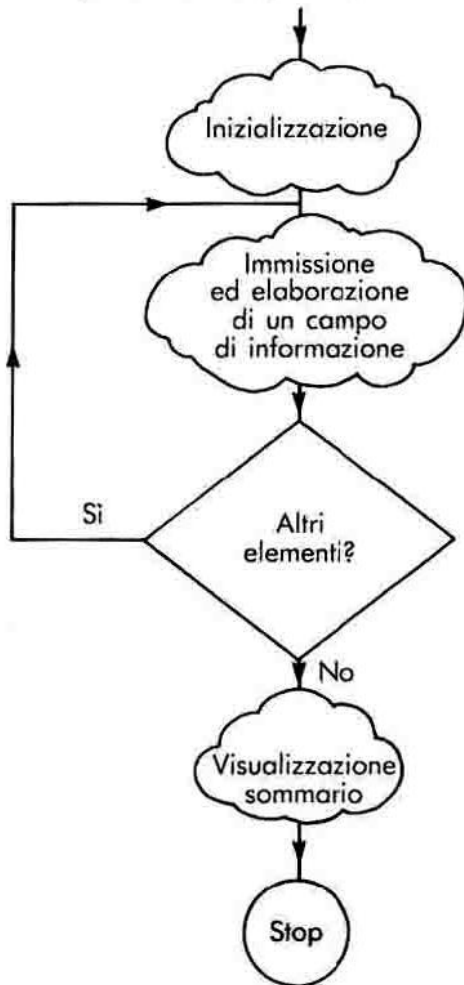
```
 10 REMCOPYRIGHT (C) ANDREW COLIN
    1982
 20 PRINT "[CLR/HOME]";
 30 PRINT "[E]" *** SOUND
    DEMONSTRATION PROGRAM ***
 40 PRINT
 50 PRINT "HIT A LETTER BETWEEN A AND M
    TO"
 60 PRINT
 70 PRINT "HEAR A SOUND EFFECT"
 80 POKE 54296, 15: POKE54295,0
100 PRINT
110 PRINT "A: HEAVY BREATHING"
120 PRINT "B: HEARTBEAT"
130 PRINT "C: PIP"
140 PRINT "D: RASPBERRY"
150 PRINT "E: WOLF WHISTLE"
160 PRINT "F: UFO VISIT"
170 PRINT "G: EXPLOSION"
180 PRINT "H: GNATS"
190 PRINT "I: POLICE SIREN"
200 PRINT "J: MANDOLIN - RISING SCALE"
210 PRINT "K: VIOLIN - FALLING SCALE"
220 PRINT "L: BUGLE CALL"
230 PRINT "M: TRIANGLE"
300 GET A$: IF A$ ="" THEN 300
310 G=ASC (A$)-64
320 IF G<1 OR G>13 THEN 300
330 GG=1024+201+40*G
340 POKE GG, PEEK (GG) + 128
```

```
350 ONGGOSUB 1000,2000,3000,4000,
    5000,6000,7000,8000,9000,10000,
    11000,12000,13000
360 POKEGG, PEEK (GG) AND 127
370 GOTO300
1000 REM ★★★ HEAVY BREATHING ★★★
1010 VV=54272
1020 POKE VV+6,0
1030 FOR MM=1 TO 10
1040 POKE VV+1,24: POKE VV+5, 138
1050 POKE VV+4,129: FORNN=1 TO 500-
    25★MM: NEXTNN
1060 POKE VV+4,0: FORNN=1 TO 100:
    NEXT NN
1070 POKE VV+1,20: POKE VV+5,155
1080 POKE VV+4, 129: FORNN = 1 TO
    1500-75★MM: NEXT NN
1090 POKE VV+4,0: FORNN= 1 TO 200:
    NEXT NN
1100 NEXTMM
1110 REM ★★★ END OF HEAVY BREATHING
    ★★★
1120 RETURN
2000 REM ★★★ HEART BEATING ★★★
2010 VV= 54272
2020 POKE VV+ 24,31: POKE VV+6,0: POKE
    VV+22,10: POKE VV+23,1
2030 FOR MM=1 TO 20
2040 POKE VV+1,24: POKE VV+5,51
2050 POKE VV+4,129: FORNN= 1 TO 50
    -MM: NEXTNN
2060 POKE VV+4,0: FORNN = 1 TO 100:
    NEXT NN
2070 POKE VV+1,16: POKE VV+5,21
2080 POKE VV+4,129: FORNN= 1 TO
    400-10★MM: NEXT NN
2090 POKE VV+4,0: FORNN= 1TO 200:
    NEXT NN
2100 NEXTMM
2110 POKE VV+24,15: POKE VV+23,0
2120 REM ★★★ END OF HEART BEATING
    ★★★
2130 RETURN
3000 REM ★★★ PIP ★★★
3010 VV=54272
3020 POKE VV+6,0: POKE VV+5,31
3030 POKE VV+1,180: POKE VV+4,33
3040 FORNN =1 TO 100: NEXT NN
3050 POKE VV +4,0
3060 REM ★★★ END OF PIP ★★★
3070 RETURN
4000 REM ★★★ RASPBERRY ★★★
4010 VV=54272: POKE VV+6,240
4020 POKE VV+1,4: POKE VV+5,0: POKE
    VV+4,33
4030 FORNN=1024 TO 512 STEP-8
4040 POKE VV+1, NN/256: POKE VV, NN
    AND 255
4050 NEXTNN
4060 POKE VV+4,0
4070 REM ★★★ END OF RASPBERRY ★★★
4080 RETURN
5000 REM ★★★ WOLF WHISTLE ★★★
5010 VV=54272
5020 POKE VV+6,0: POKE VV+5,9
5030 POKE VV+4,17
5040 FOR NN=80 TO 140STEP 4: POKE
    VV+1,NN: NEXTNN
5050 POKE VV+4,0: POKE VV+5,74
5060 FOR NN= 1TO 120: NEXTNN
5070 POKE VV+4,17
5080 FOR NN = 100 TO 120: POKE
    VV+1,NN: NEXT NN
5090 FOR NN= 120TO 60 STEP-0.4: POKE
    VV+1,NN: NEXT NN
5100 POKE VV+4,0
5110 REM ★★★ END OF WOLF WHISTLE ★★★
5120 RETURN
6000 REM ★★★ UFO VISIT ★★★
6010 VV=54272: POKE VV+1,40
6020 POKE VV+6,0: POKE VV+5,204: POKE
    VV+4,0: POKE+4,129
6030 FORNN =0 TO 2500: NEXTNN
6040 POKE VV+6,240: POKE VV+5,0: POKE
    VV+4,0
6050 POKE VV+4,17
6060 FOR MM=20 TO 6 STEP -1
6070 FOR NN= MMTO50-MM
6080 POKEVV+1,NN
6090 NEXT NN, MM
6100 POKE VV+1,3: POKE VV+4,0
6110 FOR NN= 1 TO 10
6120 POKE VV+4, 33: FOR MM= 1 TO 50:
    NEXTMM
6130 POKE VV+4,0: FOR MM=1 TO 200:
    NEXTMM
6140 NEXTNN
6150 POKE VV+6,0: POKEVV+5,5: FOR
    NN=3 TO 100 STEP 5
6160 POKE VV + 1, NN: POKE VV + 4,33
6170 FOR MM= 1TO 40 -0.25 ★ NN: NEXT
    MM
6180 POKE VV+4,0: FOR MM=1 TO 120-NN:
    NEXTMM
6190 NEXT NN
6200 POKE VV+5,31: POKE VV+4, 129
6210 FORNN=40 TO 200 STEP 0.1
6220 POKE VV+1,NN
6230 NEXT NN
6240 POKE VV+4,0
6250 REM ★★★ END OF UFO VISIT ★★★
6260 RETURN
6500 RETURN
7000 REM ★★★ EXPLOSION ★★★
7010 VV=54272: POKE VV+6,0: POKE
    VV+5,12
7020 POKE VV+1,20: POKE VV+4,129
7030 FORNN = 1 TO 2500: NEXTNN
7040 POKE VV+4,0
7050 REM ★★★ END OF EXPLOSION ★★★
7060 RETURN
7070 POKE VV+4,0
7090 RETURN
8000 REM ★★★ GNATS ★★★
8010 VV=54272
8020 POKE VV+6,0: POKE VV+5,205
8030 POKE VV+1,48+2★RND(0)
8040 POKE VV+4,33
8050 FORNN=1 TO 200: POKE VV+1,
    48+INT (2★RND (0))
8055 FORMM = 1 TO 10: NEXTMM
8060 NEXTNN
```

```
8070 POKE VV+4,0
8090 RETURN
9000 REM *** POLICE SIREN ***
9010 VV=54272: POKE VV+6,0: POKE
     VV+5,237
9020 POKE VV+2,200: POKEVV+3,0
9025 AA=55: BB=42
9030 POKE VV+1,60: POKE VV+4,65
9040 FOR NN= 1 TO 12
9050 POKE VV+1,AA
9060 FORMM = 1 TO 300: NEXTMM
9070 POKE VV+1, BB
9080 FORMM= 1 TO 300: NEXTMM
9085 IF NN=6THEN AA=53: BB=40
9090 NEXT NN
9100 POKE VV+4,0
9110 REM *** END OF POLICE SIREN ***
9120 RETURN
10000 REM *** MANDOLIN ***
10010 VV=54272: POKE VV+6,0: POKE
      VV+5,8
10020 PP=1000: QQ=2 ↑ (1/12)
10030 FOR KK=0TO4
10040 FOR JJ= 0TO 11
10050 IF JJ=1OR JJ=3 OR JJ=6 OR JJ=8 OR
      JJ=10 THEN 10080
10060POKE VV+1, PP/256: POKE VV, PP AND
      255
10070 POKEVV+4,0: POKEVV+4,33:
      FORNN=1 TO 50: NEXTNN
10080 PP=PP*QQ
10090 IFKK=4 THEN 10110
10100 NEXT JJ,KK
10110 FORNN=1 TO 200: NEXTNN
10120 POKE VV+4,0
10130 REM *** END OF MANDOLIN ***
10140 RETURN
11000 REM *** VIOLIN ***
11010 VV=54272: POKE VV+6,32: POKE
      VV+5,106
11020 PP=20000: QQ=2 ↑ (1/12)
11030 POKE VV+3,50
11040 FOR KK=1 TO 3
11050 FOR JJ =12 TO 1 STEP-1
11060 IF JJ= 1 OR JJ = 4 OR JJ = 6 OR JJ = 9
      OR JJ = 10 THEN 11150
11070 POKE VV+4,0: POKE VV+4,65
11080 FOR NN= 1TO 4
11090 ZZ=1.005*PP: POKE VV+1,ZZ/256:
      POKE VV,ZZ AND 255
11100 FORMM=1 TO 25: NEXTMM
11120 ZZ=0.995*PP: POKE VV+1, ZZ/
      256:POKE VV, ZZ AND 255
11130 FORMM=1 TO 25: NEXTMM
11140 NEXT NN
11150 PP=PP/QQ
11160 IFKK=3 THEN 11180
11170 NEXT JJ,KK
11180 FORMM= 1TO 500: NEXTMM: POKE
      VV+4,0
11190 REM *** END OF VIOLIN ***
11200 RETURN
11300 FORMM=1 TO 50: NEXTMM
12000 REM *** BUGLE CALL ***
12010 RESTORE
12020 VV=54272
12030 POKE VV+6,0: POKE VV+5,45
12040 POKE VV+3,99: PP=2500
12050 READ WW: IF WW=0 THEN POKE
      54276,0: RETURN
12060 RR=INT(WW/10):QQ=(WW-10
      *RR)*PP
12070 POKE 54273, QQ/256: POKE 54272,
      QQ AND 255
12080 POKE 54276,0: POKE 54276,65:
      FORNN= 1 TO 70 * (RR+1): NEXTNN
12090 GOTO 12050
12100 DATA 25, 6, 24, 5, 3, 3, 3, 4, 5, 6, 25, 6, 24,
      5,3,3,3, 64, 3, 3, 5, 4, 4, 5, 3, 3, 5, 4, 5, 6
12110 DATA 15, 6, 24, 5, 3, 3, 3, 4, 5, 6, 25, 6, 24,
      5,3,3,3, 94, 0
12120 REM *** END OF BUGLE CALL ***
13000 REM*** TRIANGLE ***
13010 VV=54272
13020 POKE VV+6,0: POKE VV + 5,12: POKE
      VV+1,200
13030 POKE VV+3,40
13040 FOR NN= 1 TO 5
13050 POKE VV+4,0: POKE VV+4, 65
13060 FORMM=1 TO 500: NEXTMM
13070 NEXTNN
13080 POKE VV+4,0
13090 REM *** END OF TRIANGLE ***
13100 RETURN
```

Experiment 13.1 Completed	

UNITA':14

In questa unità si esaminerà un'applicazione di computer estremamente comune: un'applicazione in cui si fa in modo che la macchina immagazzini ed elabori una grande quantità di elementi separati d'informazione e visualizzi un sommario dei risultati. Per esempio, se si volesse tener nota del proprio conto bancario si potrebbero introdurre i dettagli di ogni assegno emesso e di ogni credito pagato alla banca e la macchina potrebbe dare il saldo al termine della settimana. Per dare un altro esempio, l'insegnante potrebbe immettere nel computer i voti degli esami dei vari allievi e il computer potrebbe presentare la media totale.
Tutti i programmi di questo tipo si adeguano allo stesso profilo base, che prevede uno schema di flusso più o meno di questo tipo:

Un esempio molto semplice è dato da questo programma che immette 10 numeri e ne calcola la media:

```
10 S=0             } Inizializza
20 P=1
30 INPUT X         } Legge ed elabora
40 S=S+X             un elemento
50 P=P+1           } Altri elementi?
60 IF P < 11 THEN 30
70 PRINT"MEDIA="; S/10  } Visualizza
                          il sommario
80 STOP
```

Glossario

S: Usato per sommare i valori degli elementi
P: Usato per contare gli elementi
X: Usato per immettere i singoli elementi

Se non si comprende come funziona questo programma, controllarlo con i valori di input 3, 6, 2, 7, 0, 9, 8, 3, 12, 10.
In questo esempio è stato usato un IF-THEN per il controllo dell'iterazione, per rendere più chiara la costruzione del programma. In pratica, si scriverà il programma con un FOR...NEXT come questo:

```
10 S = 0
20 FOR P = 1 TO 10
30 INPUT X
40 S = S+X
50 NEXT P
60 PRINT"LA MEDIA È"; S/10
70 STOP
```

Si esaminerà ora la parte del programma che dice "altri elementi". Nel primo esempio alla domanda veniva risposto tenendo un semplice conteggio e usando la condizione di P < 11, che era vera fino a che non veniva immesso il decimo elemento e sommato al totale mobile. Questo metodo dipende dal fatto che il programmatore conosca in anticipo quanti elementi devono entrare nel calcolo. Il metodo è pressoché inutile in pratica in quanto non è flessibile: occorrerebbero cioè diversi programmi per trovare la media di 11 o di 20 o di qualsiasi altra serie di numeri.

È possibile scrivere un programma molto migliore se si suppone che l'utente possa dire al computer quanti elementi deve aspettarsi. Il seguente programma funzionerà per qualsiasi numero di elementi:

```
10 S = 0
20 INPUT"QUANTI NUMERI"; N
30 FOR P = 1 TO N
40 INPUT X
50 S = S + X
60 NEXT P
70 PRINT "LA MEDIA È"; S/N
80 STOP
```

Notare l'uso di N invece di 10

Glossario

S: È usato per sommare il valore degli elementi

P: È usato per contare gli elementi

X: È usato per immettere i singoli elementi

N: È usato per contenere il *numero* degli elementi

Finora ci siamo mossi su un terreno familiare, ma cosa succede quando l'utente deve immettere un grande numero di elementi (ad esempio un migliaio o più?). È illogico fargli contare gli elementi in anticipo ed irrealistico supporre che egli inserisca il numero esatto.
Un modo diverso per controllare una iterazione non consiste nell'usare un conteggio predeterminato, ma semplicemente dire al computer quando è terminato il flusso degli elementi. Si potrebbe ad esempio fare in modo che l'utente risponda alla domanda "altri elementi" ogni volta che viene eseguita l'iterazione. Ciò darebbe luogo ad un programma del tipo:

```
10 S = 0
20 N = 0
30 INPUT "PROSSIMO NUMERO"; X
40 S = S + X
50 N = N + 1
60 INPUT "ALTRI NUMERI"; M$
70 IF M$ = "SI" THEN 30
80 PRINT "LA MEDIA È"; S/N
90 STOP
```

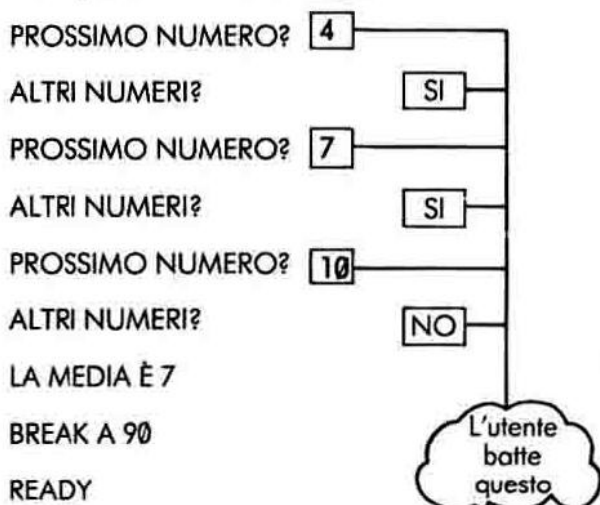
Glossario:

S: È usato per sommare il valore degli elementi

N: È usato per contare gli elementi

X: È usato per immettere i singoli elementi

M$: È usato per contenere la risposta alla domanda "Altri elementi"

Se si eseguisse questo programma, la visualizzazione potrebbe essere del tipo:

Gli inconvenienti di questo schema sono chiari. L'utente sfortunato continuerà a battere SI dopo ogni numero salvo l'ultimo. Ciò raddoppia il tempo e raddoppia il rischio di errori. Un mezzo migliore consiste nel contrassegnare la fine del flusso di elementi con un valore speciale detto *terminatore*. Una buona scelta per un terminatore è un valore che non può verificarsi come uno degli elementi. Per esempio, se si volesse usare il programma per calcolare la media dei punteggi di calcio, si potrebbe usare il numero 1000000, in quanto si è sicuri che nessuna squadra può segnare un milione di reti in una partita.

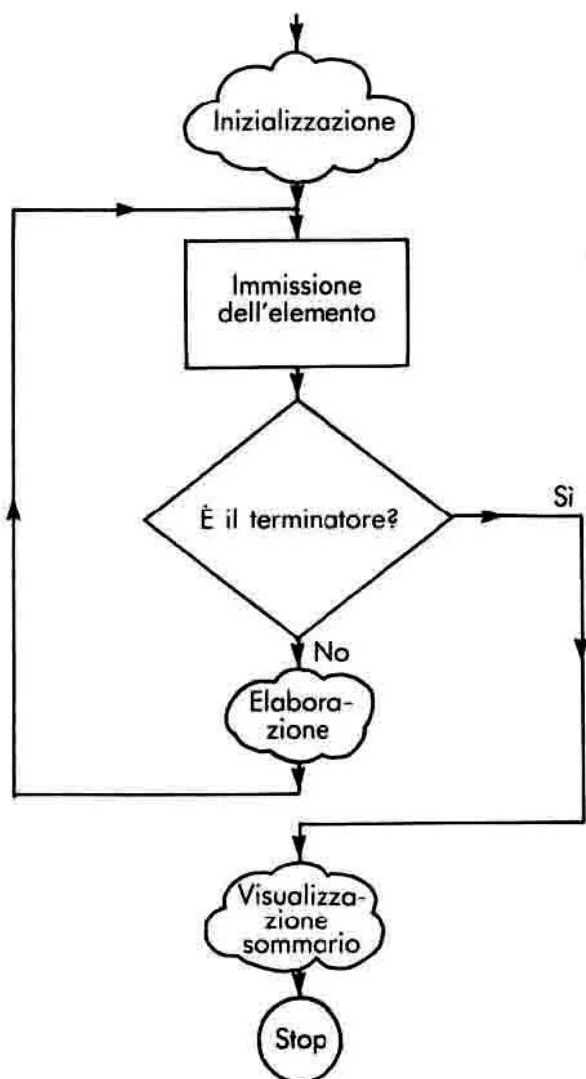
La visualizzazione prodotta da un programma scritto su queste righe potrebbe essere:

USARE 1000000 PER

TERMINARE INPUT

MEDIA 3

BREAK...

Per usare questo sistema dobbiamo ridisporre l'intero schema di flusso, in particolare la domanda "altri dati" deve venire *prima* del blocco che elabora ciascun elemento di dati — altrimenti il valore che pone termine verrebbe trattato come un elemento normale e comprometterebbe il sommario.

Il corrispondente programma per trovare una media è abbastanza lineare:

```
 10 PRINT "USARE 1000000 PER"
 20 PRINT "TERMINARE INPUT"
 30 S=0
 40 N=0
 50 INPUT "PROSSIMO NUMERO"; X
 60 IF X = 1000000 THEN 100
 70 S=S+X
 80 N=N+1
 90 GOTO 50
100 PRINT "MEDIA="; S/N
110 STOP
```

Glossario

S: È usato per sommare i valori degli elementi

N: È usato per contare gli elementi

X: È usato per immettere i singoli elementi

Per riassumere, sono stati esaminati quattro diversi modi per indicare quanti elementi d'informazione devono essere immessi da un programma. Essi sono:

1. Il numero degli elementi è specificato dal programmatore. È usato solo dai principianti e in pratica è inutile.

2. Il numero degli elementi è specificato in anticipo dall'utente. Un buon metodo se ci sono 20 elementi o meno.

3. L'utente indica dopo ciascun elemento se ce ne sono altri. Intollerabilmente noioso.

4. Il flusso di elementi termina con un valore speciale. Un buon metodo, generalmente migliore degli altri.

Scrivere un semplice programma bancario che immette il vecchio saldo e i dettagli di tutti gli assegni staccati e quindi visualizza il nuovo saldo o lo scoperto. Usare lo zero come terminatore in quanto non si scriverà mai un assegno di 0 lire. Non preoccuparsi dei crediti. Disegnare il programma in modo che possa produrre una delle due visualizzazioni che seguono:

(a) VECCHIO SALDO? 550000

BATTI DETTAGLI

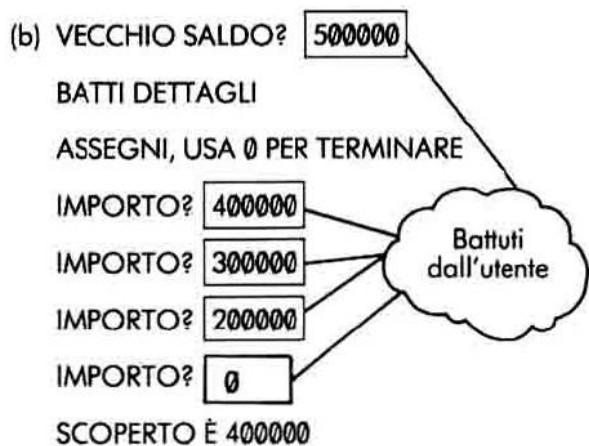
ASSEGNI, USA 0 PER TERMINARE

IMPORTO? 100000

IMPORTO? 200000

IMPORTO? 0

SALDO 250000

Battuti dall'utente

Suggerimento: La sezione relativa alla visualizzazione sarà più complessa del solito. Se B è una variabile che dà il saldo corrente, sarà negativa (o minore di 0) se si preleva più del saldo disponibile. La giusta condizione per controllare questa possibilità è $B < 0$. La soluzione dovrebbe comprendere uno schema di flusso e un glossario. Controllare a fronte della risposta l'Appendice B.

Esperimento 14.1 completato	

In alcuni problemi vari elementi nel flusso devono essere trattati in modi diversi. I corrispondenti programmi generalmente hanno dei comandi IF all'interno delle iterazioni principali. Per esempio, si supponga che dopo una mano di nera sfortuna si abbia il sospetto che una monetina fosse truccata in modo da dare testa più spesso di croce. Si potrebbe verificare questo sospetto lanciando la monetina un grande numero di volte e contando il numero di teste e croci uscite, servendosi del **64** per aiutare a tenere il conteggio dei risultati. Si potrebbe scrivere un programma in grado di dare una visualizzazione di questo tipo:

BATTI H PER TESTA

T PER CROCE

E PER TERMINARE

PROSSIMO LANCIO? H

PROSSIMO LANCIO? H

PROSSIMO LANCIO? T

. . . . e così via per 547 righe

PROSSIMO LANCIO? E

SU 547 LANCI

ERANO 490 TESTE

E 57 CROCI

READY.

(Battuti dall'utente)

In questo modo si potrebbero ricavare le proprie conclusioni sul trucco della monetina.
Disegnare e scrivere questo programma, dal glossario e dallo schema di flusso fino ai comandi BASIC.
La visualizzazione campione mostra che si usa un valore speciale E per terminare il flusso di dati. Lo schema di flusso sarà più o meno come quello di pagina 109 e tutto ciò che occorre fare è espandere le nuvolette.

H: Per contare i numeri delle teste

T: Per contare il numero delle croci

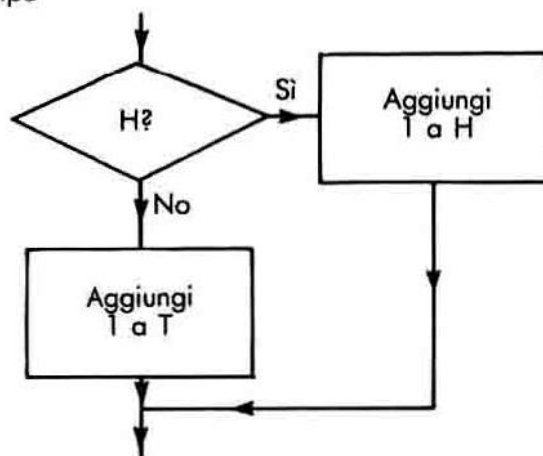
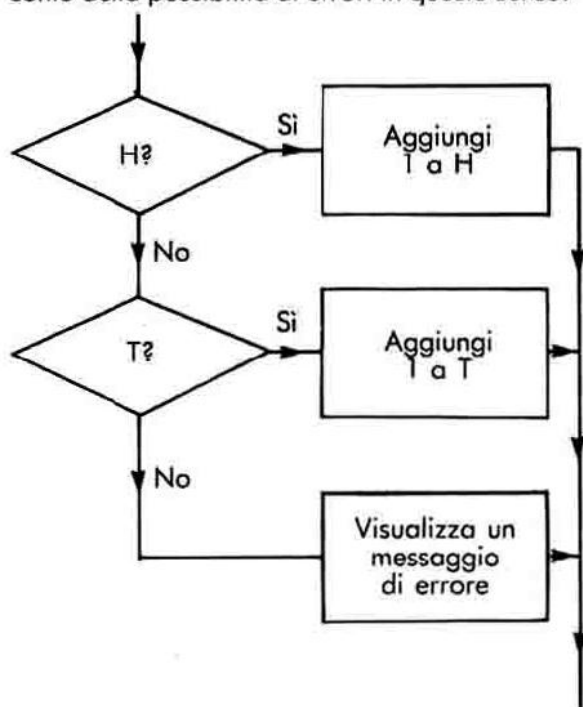
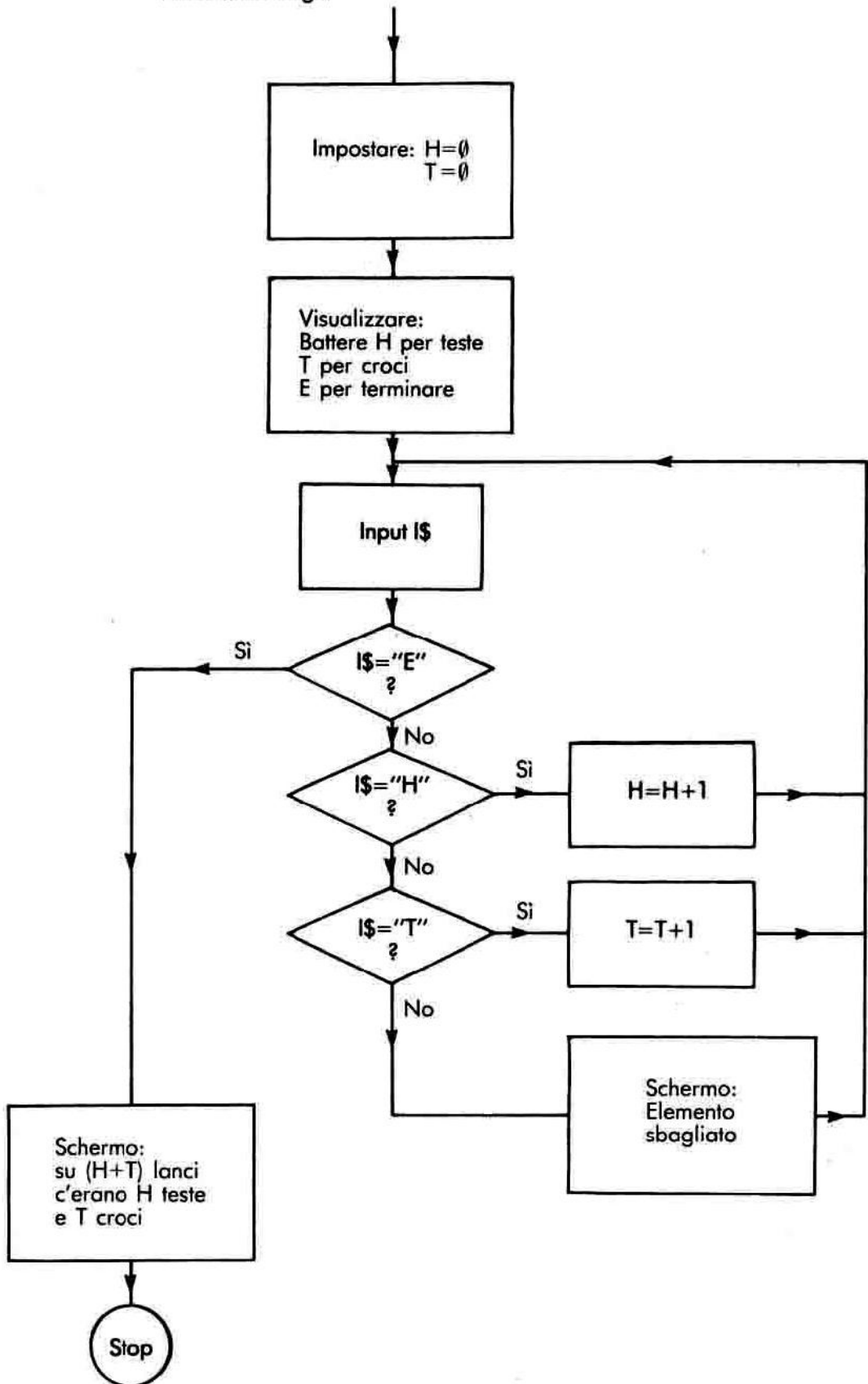
I$: Per immettere un elemento

(Come al solito, i nomi H e T sono scelti liberamente).

Alcuni potrebbero essere tentati di comprendere una quarta variabile per contare il numero totale dei lanci, ma sarebbe un lavoro inutile; il totale è sempre indicato dall'espressione H + T (numero delle teste + numero delle croci).

Successivamente è possibile elaborare la sezione inizializzazione del programma. Ci sono due cose da fare:

- Impostare le variabili H e T a zero
- Visualizzare il messaggio di testa

Si passerà poi alla nuvoletta all'interno dell'iterazione principale che elabora ciascun nuovo elemento. In questa fase la "E" sarà stata filtrata e ogni elemento sarà una H o una T. Il lavoro base che la nuvoletta deve fare è di aggiungere 1 al totale delle teste o al totale delle croci. Un approccio possibile consisterebbe nell'usare l'argomento "è un'H? se no deve essere una T". Ciò darebbe luogo ad uno schema di flusso del tipo

In pratica questo metodo non verrebbe mai usato da un buon programmatore in quanto non terrebbe conto degli errori di battitura dell'utente. Se l'utente batte una J invece di una H (che si trovano vicine l'una all'altra sulla tastiera), il programma la conterebbe come T, che non è certo ciò che l'utente desidera. È molto meglio tenere conto della possibilità di errori in questo senso:

Un programma che consente all'utente di fare errori senza conseguenze disastrose è detto "robusto".
Infine, si può espandere la nuvoletta di sommario per dare un prospetto in tre righe al termine della visualizzazione. Lo schema ampliato apparirebbe come segue:

Qui di seguito è riportato il corrispondente programma. Notare che l'iterazione principale è un pochino aggrovigliata. Ciò è inevitabile dato che dobbiamo forzare uno schema di flusso bidimensionale in una singola serie di istruzioni.

```
10 H=0
20 T=0
30 PRINT "BATTI H PER TESTE"
40 PRINT "T PER CROCI"
50 PRINT "E PER TERMINARE"
60 INPUT "PROSSIMO LANCIO"; I$
70 IF I$="E" THEN 160
80 IF I$="H" THEN 120
90 IF I$="T" THEN 140
100 PRINT "ELEMENTO SBAGLIATO"
110 GOTO 60
120 H=H+1
130 GOTO 60
140 T=T+1
150 GOTO 60
160 PRINT"SU"; H+T;"LANCI"
170 PRINT"C'ERANO"; H;"TESTE"
180 PRINT"E"; T;"CROCI"
190 STOP
```


ESPERIMENTO 14·2

(a) Se un programma ha molti input, l'utente può anche non osservare lo schermo mentre batte. È una buona idea fare in modo che il programma reagisca con suoni nonchè con messaggi visualizzati. Si potrebbe per esempio usare un grazioso "pip" per un elemento accettabile ed un rude rumore per quello che non lo è. Osservare il programma delle teste e delle croci. Ogni volta che l'utente batte una H la macchina esegue i comandi alle righe 120 e 130. Sarebbe possibile inserire un adatto rumore aggiungendo i comandi:

```
121 POKE 54272 + 24, 15
122 POKE 54276,0 : POKE 54278,255:
    POKE 54277,0 : POKE 54272,90:
    POKE 54273,150
123 POKE 54276,17
124 FOR M = 1 TO 100
125 NEXT M
126 POKE 54276,0
```

Caricare il programma HEADS dalla cassetta di nastro e correggerlo in modo che risponda a ciascun input (giusto o sbagliato) con un suono adatto.

(b) Una volta gli orologi erano soggetti ad una curiosa forma di tassa che era calcolata come segue: se il prezzo dell'orologio era minore di 12 sterline, la tassa era pari ad ⅓ del costo.
Se il prezzo era compreso fra 12 e 16 sterline, la tassa era di 4 sterline.

Se il prezzo superava le 16 sterline, la tassa era pari a 1/4 del costo dell'orologio.
Scrivere un programma che immetta un'elenco dei prezzi degli orologi terminato da 0 e che visualizzi i totali da addebitare per ciascun orologio (compreso il costo della tassa). Notare che questo programma avrà uno o più comandi PRINT all'interno dell'iterazione e non ha bisogno di un blocco di sommario. Si troverà indispensabile un buon schema di flusso.

(c) Scrivere un programma che immetta un flusso di numeri terminati da 0 e che visualizzi il maggiore di essi.
Suggerimento: usare una variabile per registrare il numero maggiore in qualsiasi momento e aggiornarlo ogni volta che viene eseguita l'iterazione.

Esperimento 14.2 completato	

Controllare le risposte nell'appendice B.

UNITA':15

Questa unità tratta tre importanti caratteristiche del BASIC Commodore che sono utili nei giochi, nei quiz e in altri programmi dove la macchina e il suo utente lavorano a stretto contatto.
Si inizierà dando uno sguardo a "REACTION" uno dei programmi che si troverà sulla cassetta di nastro. Il "tempo di reazione è una misura della rapidità con la quale la persona risponde ad un evento inaspettato. Un guidatore sicuro deve avere un breve tempo di reazione in modo da poter azionare i freni rapidamente quando un bambino attraversa la strada davanti alla sua vettura. Un buon tempo di reazione è anche utile nella maggior parte degli sport e in molte professioni.
La maggior parte delle persone, se fanno attenzione, hanno tempi di reazione fra 0,2 e 0,3 secondi (da 20 a 30 centesimi di secondo). Un tempo di reazione minore di 0,2 fa pensare ad una persona particolarmente sveglia mentre un tempo di reazione di più di 0,3 secondi è tipico di chi ha bevuto qualche bicchiere di troppo.

ESPERIMENTO 15·1

Caricare il programma REACTION e usarlo per misurare il proprio tempo di reazione. Eseguire il programma parecchie volte e ignorare i primi due o tre risultati, dato che non saranno tipici. Continuare a provare il programma fino a che non lo si è compreso a fondo e lo si può usare con fiducia per misurare il tempo di reazione di un amico che non ha conoscenza di programmazione.
È possibile notare tre aspetti del programma che non sono immediatamente ovvii:
Primo: quando le istruzioni dicono "qualsiasi tasto" intendono realmente ciò. Si troverà che i tasti di funzionamento tipo RETURN e INST DEL lavorano altrettanto bene delle lettere o dei numeri.
Secondo: il tempo che occorre lasciar trascorrere prima di udire il tono è sempre diverso: esso varia da 1 a 6 secondi in un modo che non è possibile prevederlo in anticipo.
Terzo: se si preme un tasto prima che inizi il tono, si ottiene il messaggio "TOO SOON" (troppo presto).

Si esaminerà ora il programma in dettaglio e si spiegherà come funziona. Iniziamo con l'esaminare lo schema di flusso e il programma BASIC, indicati qui di seguito.

Nuvoletta 1

```
10 REM REACTION TIME PROGRAM
20 PRINT " [SHIFT] and [CLR HOME] "
30 PRINT "TO MEASURE YOUR"
40 PRINT "REACTION TIME:"
50 PRINT "HIT ANY KEY"
60 PRINT "THEN WAIT FOR THE"
70 PRINT "TONE. WHEN YOU"
80 PRINT "HEAR IT, HIT ANY"
90 PRINT "KEY AS FAST AS"
100 PRINT "YOU CAN. GOOD LUCK!"
```

Nuvoletta 2

```
110 REM WAIT FOR ANY KEY
120 GET A$
130 IF A$ = " " THEN 120
```

Nuvoletta 3 (e vedere più avanti)

```
140 REM WAIT A RANDOM TIME
143 PRINT
145 PRINT "WAIT FOR IT!"
148 PRINT
150 Q=TI+INT(60+301*RND(0))
160 GET A$
170 IF A$<>" " THEN 340
180 IF TI<Q THEN 160
```

Nuvoletta 4

```
190 REM START TONE AND NOTE TIME
200 POKE 54295,0
202 POKE 54296,15
205 POKE 54278,240
207 POKE 54277,16
210 POKE 54273,100
212 POKE 54276,33
220 X = TI
```

Nuvoletta 5

```
230 REM WAIT FOR ANY KEY
240 GET A$
250 IF A$= " " THEN 240
```

Nuvoletta 6

```
260 REM GET RESULT AND STOP TONE
270 R=TI
280 POKE 54276,0
290 POKE 54296,0
```

Nuvoletta 7

```
300 REM DISPLAY RESULT
310 PRINT "YOUR REACTION TIME IS"
320 PRINT (R—X)/60 ; "SECONDS"
```

```
330 STOP
```

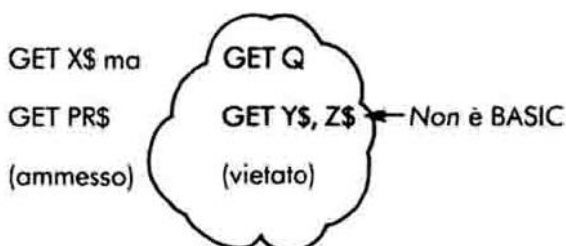
Parte della nuvoletta 3

```
340 PRINT "TOO SOON"
350 STOP
```

Il programma è stato contrassegnato in modo che siano chiaramente visibili i comandi che corrispondono a ciascuna nuvoletta nello schema di flusso.
La prima nuvoletta (righe da 10 a 100) si compone interamente di comandi PRINT ed è abbastanza lineare.
La seconda nuvoletta, righe da 110 a 130, fa sì che il programma attenda fino a che l'utente non batte il tasto. La nuvoletta N. 2 usa un comando con una nuova parola chiave:

GET A$

Questo comando è in qualche modo identico a INPUT e trasferisce le informazioni dalla tastiera al computer. In ogni caso ci sono alcune importanti differenze:

1. La parola chiave GET deve essere seguita esattamente da un nome variabile stringa. I nomi di variabili numeriche non sono ammessi. Per esempio:

2. Il comando GET non attende che l'utente faccia qualcosa, ma esamina semplicemente la tastiera in quell'istante e indica quale tasto è stato battuto dall'ultimo comando GET o INPUT. Se è stato battuto un tasto, viene trasformato in una stringa ad un carattere e inserito nella variabile citata nel comando GET. Se non è stato di recente battuto un tasto, la variabile viene impostata ad una stringa nulla. Si tratta di una stringa senza caratteri, normalmente scritta nella forma" ".
   Per illustrare questa regola si immagini di avviare il computer sul seguente programma con iterazioni e osservare cosa succede all'interno della macchina:

```
10 GET X$
20 GOTO 10
```

   Il computer esegue questa iterazione per circa 50 volte in un secondo. Fintantoché l'utente non tocca la tastiera, X$ viene impostato ad una stringa nulla e cioè: " ".
   Si supponga ora che l'utente prema un tasto — ad esempio quello contrassegnato U. Non appena viene eseguito il comando GET (e cioè entro un 50esimo di secondo) X$ sarà impostato alla stringa "U".
   In ogni caso ciò si verifica soltanto una volta per ogni manovra sui tasti; la successiva iterazione X$ sarà di nuovo impostata a " " e ciò continuerà fino a che viene rilasciato il tasto U e viene premuto un altro tasto (o lo stesso tasto). Le sole eccezioni a questa regola sono i cosidetti tasti di ripetizione tipo il tasto di spazio.

3. Il comando GET non tratta taluni caratteri di controllo tipo INST DEL, RETURN o i comandi di controllo del cursore come tasti speciali ma li tratta allo stesso modo salvo STOP, che interrompe il programma.

4. Qualsiasi carattere che viene rilevato dal comando GET, non è visualizzato sullo schermo.
   Tenendo presente questi punti, è possibile ora iniziare a dare un certo senso alle righe 120 e 130 nel programma REACTION. Il comando 120 esamina la tastiera e fornisce una stringa in A$ che è nulla a meno che non sia stato premuto un tasto. Il comando 130 prova A$ e fa sì che il computer torni alla 120 fino a che l'utente non batte un qualsiasi tasto. A quel punto il programma può passare alla riga 140.
   Lo scopo di questa nuvoletta è di tenere il programma in attesa fino a che l'utente non mostra che è pronto a provare il suo tempo di reazione. Perchè usare un'iterazione con un GET invece di un singolo comando tipo

   INPUT "READY"; A$?

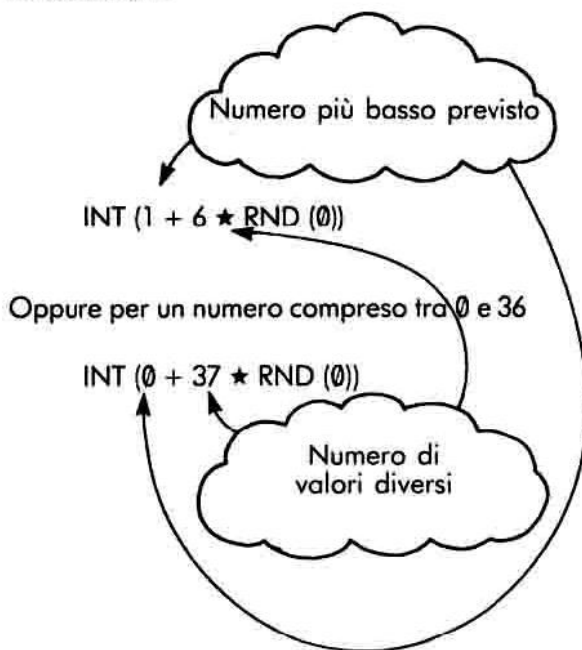
   Ci sono due motivi. Primo: INPUT aspetta sempre un RETURN dopo il messaggio dell'utente. Ciò implica un minimo di due caratteri da battere.
   Secondo: GET tratta pressochè tutti i caratteri allo stesso modo cosicchè c'è molto meno pericolo che il programma venga danneggiato se l'utente batte un tasto di funzione anzichè una lettera o un numero.
   La nuvoletta numero 3 fa sì che la macchina attenda casualmente (e cioè in maniera imprevedibile) fra il segnale di pronto dell'utente e il tono. Il tempo di attesa deve essere variabile, in quanto se fosse sempre lo stesso, l'utente imparerebbe quanto deve attendere prima dell'emissione del tono e non si tratterebbe in questo caso di un evento inaspettato.
   La nuvoletta usa due funzioni mai incontrate fino ad oggi: la funzione *casuale* e il *temporizzatore* interno.

La funzione casuale è un modo per far sì che la macchina produca un numero *imprevedibile**. Ogni volta che la macchina elabora l'espressione RDN (0) dà un valore diverso compreso fra 0 e 1.
Nella maggioranza dei casi pratici, non occorre una frazione casuale tra 0 e 1 ma un numero intero casuale entro limiti che dipendono dal problema da risolvere. Per esempio, se si fa in modo che la macchina imiti qualcuno che lanci un dado a 6 facce, ci si aspetta un numero tra 1 e 6; oppure se si vuole riprodurre una roulette (europea) occorre un numero compreso tra 0 e 36.
Per ottenere un numero intero in qualsiasi campo specificato si usa una espressione leggermente diversa:

INT (x + y ★ RND (0))

dove x è il numero più basso che occorre.

y è il *numero delle diverse possibilità*

Così per ottenere un numero da 1 a 6 occorrerebbe inserire

Il numero non è realmente imprevedibile in quanto tutto ciò che succede in un computer dipende da ciò che è successo precedentemente. In ogni caso, ciascun nuovo numero "casuale" è derivato dal precedente mediante un processo complicato di elevamento al quadrato e di manipolazione delle cifre del risultato per cui, a meno che non si conosca esattamente l'algoritmo, non si può certamente dire quale sarà il numero che uscirà successivamente.

Un'espressione di questo tipo può essere inclusa in un'iterazione in modo da poterla eseguire parecchie volte. Battere il seguente programma che imita 120 lanci di un dado:

```
NEW

10 FOR J = 1 TO 120

20 S = INT (1 + 6 ★ RND (0))

30 PRINT S;

40 NEXT J

50 STOP
```


Lanciare questo programma e contare il numero degli 1, dei 2... dei 6 che compaiono sullo schermo. Immettere i risultati nella prima fila della tabella che segue:

	1	2	3	4	5	6
Numero lanci (1)						
Numero lanci (2)						

Il programma è una buona imitazione di un gioco di dadi onesto (non truccato)?
Eseguire il programma di nuovo e compilare la seconda fila. Esaminare i risultati e notare che essi differiscono dalla prima esattamente come ci si aspetterebbe dal gioco reale.
L'altra caratteristica importante della nuvoletta 3 è il temporizzatore interno, TI. Si è già incontrato l'orologio TI$, che conta il tempo in ore, minuti e secondi, ma la variabile speciale TI (che non è un stringa ma un numero) si propone di misurare periodi di tempo più brevi. TI è impostato a 0 quando il **64** viene avviato e da quel momento in poi, qualunque cosa succeda, viene aggiunto un 1 ogni 60esimo di un secondo. Questo intervallo, 1/60 di secondo è detto un "jiffy". È possibile ottenere il valore corrente del temporizzatore interno in qualsiasi momento in jiffy usando il nome TI in un'espressione; ma non è per contro possibile variare il valore nel modo in cui è possibile farlo con TI$.
Impartire il comando

PRINT TI

La macchina risponderà visualizzando un numero abbastanza grande (60★60 o 3600 jiffies per ogni minuto durante il quale la macchina è stata accesa). Ora provare di nuovo il comando: si noterà che il valore è aumentato di qualche centinaia. Finalmente provare a ripristinare il valore di TI e vedere cosa succede!
TI può essere usato per misurare periodi di tempo in due modi diversi ma correlati. In nessuno di essi siamo interessati al numero di jiffies trascorsi da quando il **64** è stato acceso. Per contro ci serviamo del fatto che la durata di

qualsiasi intervallo di tempo è indicata dalla *differenza* tra TI al termine di tale tempo e il valore che aveva all'inizio. Per esempio, al termine di un periodo di 5 secondi, TI sarà 5 ★ 60 o 300 di più di quanto non fosse all'inizio. Ciò è vero sia che la macchina sia stata accesa per 5 secondi o per 5 anni.
Nel primo modo di uso del temporizzatore interno, si fa in modo che la macchina misuri un periodo di tempo che è noto in anticipo e la macchina dice quanto tempo è trascorso. Il metodo è semplice. All'inizio del periodo il programma osserva TI e prevede che valore avrà al termine del periodo; quindi attende in un'iterazione fino a che TI raggiunge (o supera) quel valore. È più o meno quello che si fa in cucina quando si dice "queste patate devono bollire per 20 minuti. Ora sono le 4 e 10 cosicchè dovremmo toglierle dal fuoco alle 4 e 35".
Per illustrare questo argomento c'è un programma di temporizzatore per impiego generico che si potrebbe usare in cucina, in laboratorio, ecc.

```
10 INPUT "QUANTI MINUTI": M

20 R=TI+M★3600

30 IF TI < R THEN 30

40 PRINT "TEMPO SCADUTO!"

50 STOP
```

Se si cerca di eseguire questo programma, usare un piccolo numero di minuti altrimenti si trascorre un mucchio di tempo in attesa. Quando si studia il programma, ricordarsi che TI continua ad aumentare per tutto il tempo cosicchè al limite, dopo M★3600 jiffies, la condizione TI < R potrebbe essere falsa.
Nella seconda variante, si vuole che il computer dica quanto tempo trascorre da un dato momento fino al verificarsi di qualche evento. Si fa in modo che la macchina registri il valore di TI all'inizio del periodo di calcolo. Quando l'evento si verifica, la differenza tra il valore attuale di TI e il valore registrato è una misura dell'intervallo espressa in jiffies. Questa seconda variante ricorda i semplici calcoli del montanaro quando dice: "Ricordo di aver iniziato a scalare questa collina alle 5 di questa mattina. Ho raggiunto la cima alle 11 e mi sono occorse perciò 6 ore".
Un programma che misurasse il tempo in questo modo avrebbe comandi di questo genere:

R=TI (Memorizza il valore di TI all'inizio del periodo)

e successivamente

E=TI Ottiene il valore di TI al termine del periodo

D=E-R Ricava la differenza di tempo (in jiffies)

S=D/60 Ricava la differenza di tempo (in secondi)

PRINT "THAT TOOK"; S; "SECONDS"

Ora è possibile riunire i comandi nella nuvoletta numero 3.
Si vuole attendere un periodo compreso tra 1 e 5 secondi. Il che significa tra 60 e 300 jiffies, da decidersi dalla macchina in maniera imprevedibile. L'espressione appropriata è

INT (60 + 301 ★RND (0))

Il periodo di attesa è deciso immediatamente prima dell'inizio del periodo stesso, cosicchè è noto in anticipo (quantunque non all'utente). Si usa il primo metodo di calcolo del tempo che comporta la previsione del valore di TI al termine del periodo. Il comando 150 effettua questa previsione e registra i valori in Q.
Se questo fosse tutto ciò che occorre, l'intera nuvoletta potrebbe comparire come segue:

```
150 Q=TI+INT (60+301★RND (0))

160 IF TI <Q THEN 160
```

Così com'è, occorre controllare che l'utente non batta un tasto prima di udire il tono. I comandi 160, 170, 340 e 350, sono inclusi semplicemente per controllare questa possibilità.
Il resto del programma è completamente lineare. Il valore di TI all'inizio del periodo di reazione è memorizzato in X e i comandi 240 e 250 vengono usati in modo che il programma attenda che l'utente batta un tasto.
Studiare il programma attentamente e assicurarsi di comprenderne ogni comando.

Esperimento 15.1 completato	

ESPERIMENTO 15·2

1. Scrivere un programma del "cronometro". Quando l'utente batte il tasto B, il programma inizia a contare. Quando batte S, si interrompe e visualizza il tempo trascorso in secondi.
   Il programma dovrebbe visualizzare le istruzioni in modo da poter essere usato da chiunque senza ulteriori spiegazioni.

   Suggerimento: usare GET e TI.

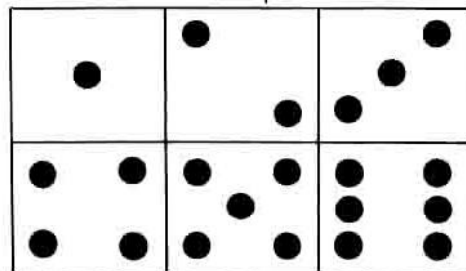
2. Scrivere un programma che imita il lancio di una monetina. Ogni volta che l'utente preme il tasto, il programma visualizza "HEADS" (testa) o "TAILS" (croce) a caso.

Esperimento 15.2 completato	

Controllare ora le risposte nell'Appendice B.

ESPERIMENTO 15·3

I numeri casuali sono utili nella programmazione dei giochi di probabilità, ad esempio il lancio dei dadi, il funzionamento di slot machine e così via. Tutti questi programmi seguono lo stesso profilo base per un "lancio" o "colpo" è il seguente:

Si illustrerà questa idea con il vecchio gioco della corona e àncora*, che si gioca con tre dadi e una tavoletta divisa in sei quadrati*:

Questo gioco assume solitamente diversi simboli ma ciò non influisce sul principio del gioco stesso.

Il giocatore punta la sua scommessa su uno qualsiasi dei quadrati. Per esempio, potrebbe scommettere 5 mila lire su . Quindi chi tiene il banco getta i tre dadi. Se uno di essi esce con il giocatore vince il doppio della sua posta; se due dadi escono con il giocatore riceve il triplo della posta originale e se il esce su tutti e tre i dadi, il giocatore è remunerato con quattro volte la sua posta. Tutte queste vincite comprendono la posta originale. Per contro, se non esce alcun il giocatore perde la posta. Qui di seguito è indicato il programma per giocare una mano di corona e áncora. Usando il glossario, non si dovrebbe avere difficoltà nel seguirlo.

S:	Posta del giocatore
N:	Numero puntato dal giocatore
D1, D2, D3	Risultato del lancio di 3 dadi
C:	Numero di dadi usciti con N, il numero del giocatore

```
10 INPUT "POSTA"; S
20 INPUT "NUMERO PUNTATO (1-6)"; N
30 D1 = INT (1+6*RND(0))
40 D2 = INT (1+6*RND(0))
50 D3 = INT (1+6*RND(0))
```

Lancio di 3 dadi

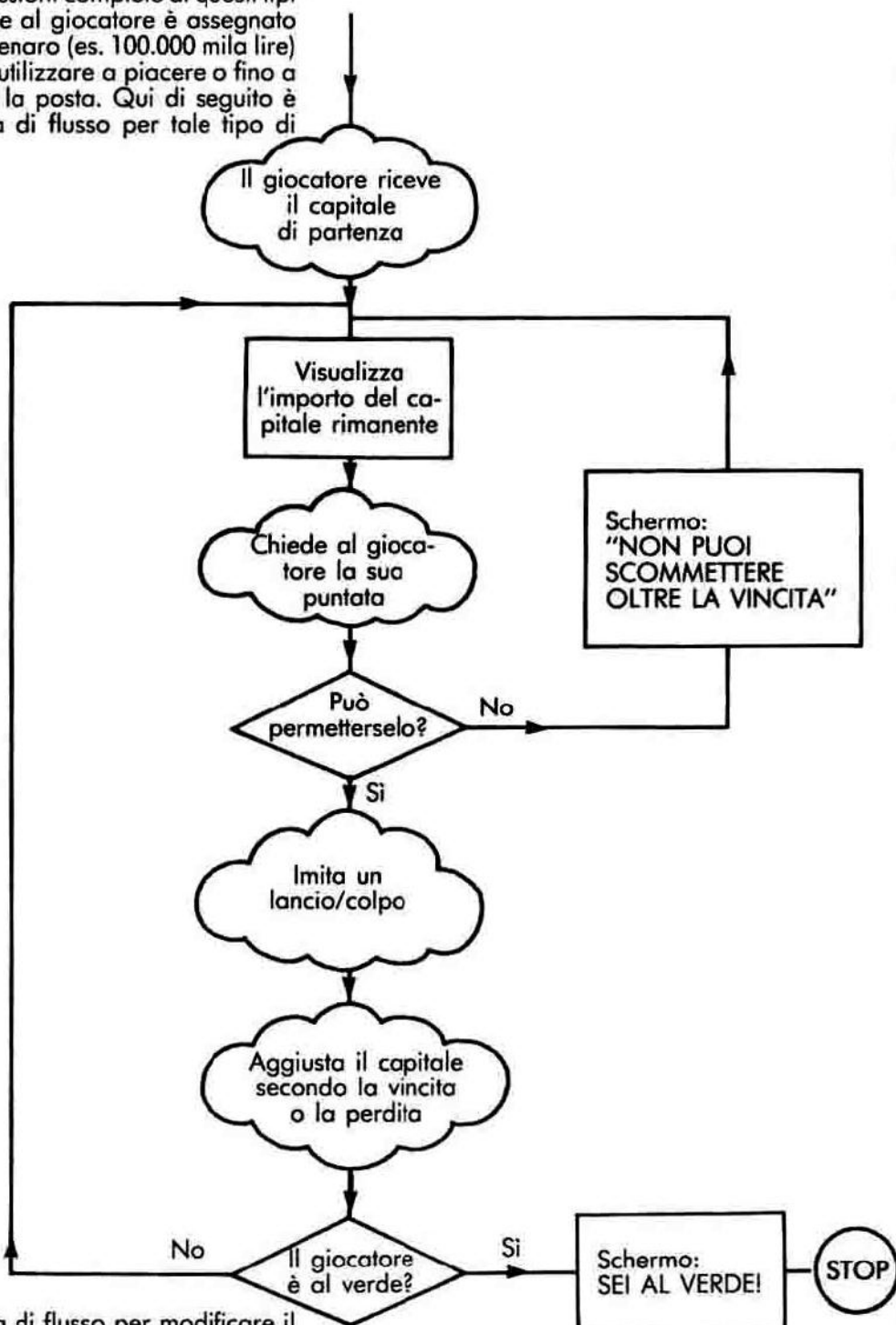
```
60 C = 0
70 IF D1 <> N THEN 90
80 C = C+1
90 IF D2 <> N THEN 110
100 C = C+1
110 IF D3 <> N THEN 130
120 C = C+1
```

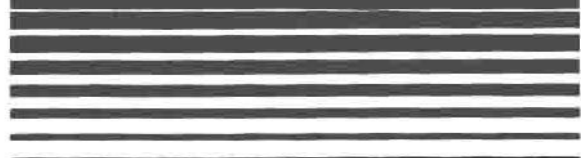
Conta il numero di dadi usciti col numero puntato dal giocatore

```
130 PRINT "RISULTATI USCITI:"; D1; D2; D3
140 IF C <> 0 THEN 170
150 PRINT "HAI PERSO"
160 GOTO 180
170 PRINT "VINCI"; S*(C+1); "LIRE"
```

Visualizza i risultati

```
180 STOP
```

Pochissimi giocatori interrompono il gioco dopo un lancio. Solitamente la gente inizia con un certo capitale e continua a giocare fino a che non vada in rovina o — molto raramente — faccia saltare il banco.
I programmi di giochi d'azzardo sul **64** sono migliori se imitano sessioni complete di questi tipi di giochi. Inizialmente al giocatore è assegnato un certo importo di denaro (es. 100.000 mila lire) che il giocatore può utilizzare a piacere o fino a che non ha esaurito la posta. Qui di seguito è indicato uno schema di flusso per tale tipo di gioco:

Usare questo schema di flusso per modificare il programa corona e áncora in modo da far partire l'utente con un capitale di 100.000 mila lire e consentirgli di giocare a piacere. Quando il programma è completo, eseguirlo parecchie volte e decidere se è meglio fungere da giocatore o da chi tiene il banco!

Esperimento 15.3 completato	

ESPERIMENTO 15·4

Scrivere un programma per imitare qualsiasi altro gioco di probabilità: gioco d'azzardo con dadi, tressette, ecc. Abbellire il programma con immagini di dadi o di carte, suoni adatti e cosi via.

Esperimento 15.4 completato	

L'Appendice B contiene un programma "craps" (dadi) da provare.

CONCLUSIONE

CONCLUSIONE

Congratulazioni per avere raggiunto la fine del corso. A questo punto avrete certamente acquisito una buona conoscenza dei principi della programmazione e sarete in grado di disegnare e scrivere programmi per una vasta gamma di problemi e di applicazioni interessanti. Speriamo che abbiate anche preso l'abitudine ad un disegno curato e preciso, alla conservazione ed all'archiviazione degli schemi di flusso, dei glossari e delle note per i programmi. Sono queste le qualità di pianificazione e di auto-organizzazione che distinguono il programmatore veramente competente dagli altri.
A questo punto avete compiuto metà della strada nello studio del BASIC. Ci sono molti altri importanti problemi che richiedono parti del linguaggio che non avete ancora studiato. Per esempio, potreste voler far muovere immagini sullo schermo o riordinare nomi di persone in ordine alfabetico oppure memorizzarli sulla cassetta di nastro. Questi argomenti e molti altri sono completamente spiegati nel secondo libro di questa serie intitolato

INTRODUZIONE AL BASIC (Parte II)

scritto nello stesso stile di quello che avete appena terminato e che completerà la vostra conoscenza del linguaggio BASIC.
La programmazione — come abbiamo detto nell'introduzione — è un campo molto vasto. Ora che avete fatto un buon inizio dovreste allargare la vostra conoscenza in tre modi:

(a) Leggendo quanto più potete. Vale la pena di leggere molte riviste sull'argomento, specialmente Commodore Computer Club o libri sulla programmazione anche se non si riferiscono specificamente al **64.**

(b) Associandovi ad un club locale di appassionati di computer. Troverete gli indirizzi su Computer Club.

(c) Lavorando ai propri programmi. Puntate alla perfezione. Progettate programmi "robusti" e utilizzabili da chiunque senza speciali istruzioni;
Scriveteli in modo da poter essere orgogliosi, non timorosi, di mostrarne il funzionamento ad altri esperti di computer.

Un ultimo punto. Avete trovato un hobby affascinante e probabilmente una professione per il futuro. Ricordate che insieme ai vantaggi della conoscenza dei computer c'è anche la responsabilità di far sì che vengano usati umanamente e in maniera saggia. Nessuno vuole una società controllata dal computer con poco lavoro e nessuna libertà. Tocca a voi — fra gli altri — evitarlo.

APPENDICI

APPENDICE A

Il **64** è un computer in grado di eseguire calcoli matematici su grande scala; a titolo puramente storico diremo che può eseguire calcoli aritmetici a velocità sensibilmente maggiore della gran parte dei computer su grande scala installati prima del 1960!
Questa Appendice sottolinea Alcune delle funzioni matematiche del **64**. Occorre soltanto leggerla e conoscere il materiale in essa contenuto se si vuole usare il computer per calcoli matematici, scientifici o tecnici. Alcune delle caratteristiche descritte sono abbastanza semplici e possono essere facilmente comprese da chiunque ricordi l'aritmetica elementare imparata a scuola. Altre caratteristiche richiedono una conoscenza più profonda, a livello scuola media superiore. Occorre soltanto procedere fino al rispettivo livello di conoscenza. Ci si aspetta comunque che siano state lette tutte le unità che compongono il corso.

1. Espressioni

Le espressioni citate per la prima volta nell'unità 4, sono esempi molto semplici di funzioni più complesse. Così nei comandi

A = 34

B = B + 1

C = ((X + Y) − 34.7/(Q−3)) ★ (Z−3) ↑2

le parti sottolineate sono tutte espressioni che il **64** elabora per conto dell'operatore.
Le espressioni sono costituite da tre tipi di elementi:

Valori: variabili numeriche o numeri tipo

B, X, Y, 34, 34.7

Operatori: i segni + − ★ / e ↑

(↑ significa "elevato alla potenza")

Parentesi: (e)

Le espressioni nel BASIC sono scritte nello stesso modo dell'algebra ordinaria e hanno lo stesso significato. Ci sono comunque quattro piccole differenze:

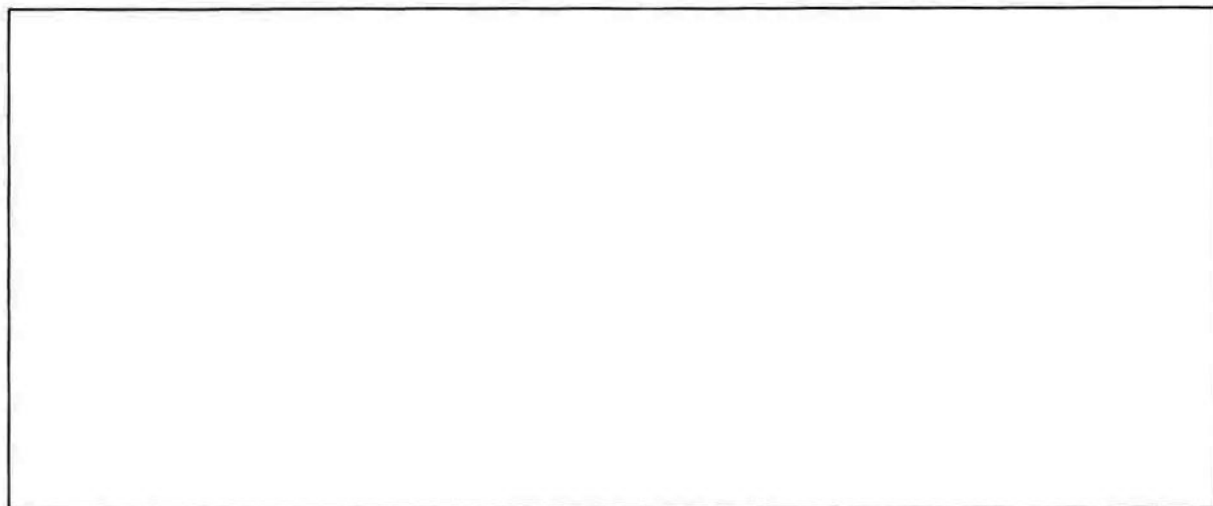
- Le espressioni BASIC sono scritte in maiuscolo anzichè in minuscolo
- L'elevamento ad esponente deve essere indicato con il segno ↑ in quanto lo schermo del **64** non consente di scrivere piccoli numeri al di sopra della riga. Invece di "3^2", occorre inserire "3↑2".
- La moltiplicazione deve sempre essere indicata usando il segno ★. In BASIC occorrerà scrivere "3★A" e non "3A" come nell'algebra convenzionale. Questa regola può essere una fonte di errori talvolta difficili da rilevare. Se si immette BA dove si intende B★A, la macchina assume che si stia parlando di una nuova variabile denominata BA. Di conseguenza non emetterà un messaggio di errore ma produrrà la risposta sbagliata!
- La divisione è scritta nella forma A/B e non $\frac{A}{B}$.
  Se il numeratore o il denominatore della frazione è un'espressione complicata occorre delimitarla con le parentesi. Il modo corretto per scrivere $\frac{3+5}{7+8}$ in BASIC è (3+5)/(7+8).
  Se si trascurano le parentesi e si scrive 3+5/7+8, le regole di precedenza (che sono indicate nel paragrafo successivo) fanno sì che la macchina tratti questa espressione come se fosse scritta $3+\frac{5}{7}+8$.

Quando il **64** elabora un'espressione, si occupa per la prima cosa del segno ↑ quindi del segno di moltiplicazione e di divisione e infine del segno di addizione e di sottrazione, lavorando da sinistra a destra in ogni caso. Tutto ciò che è contenuto tra parentesi viene calcolato per primo. Si tratta delle cosiddette regole di precedenza, che danno gli stessi risultati della normale algebra scolastica.
Il valore dei numeri nelle espressioni non deve essere necessariamente intero, ma può anche essere decimale.

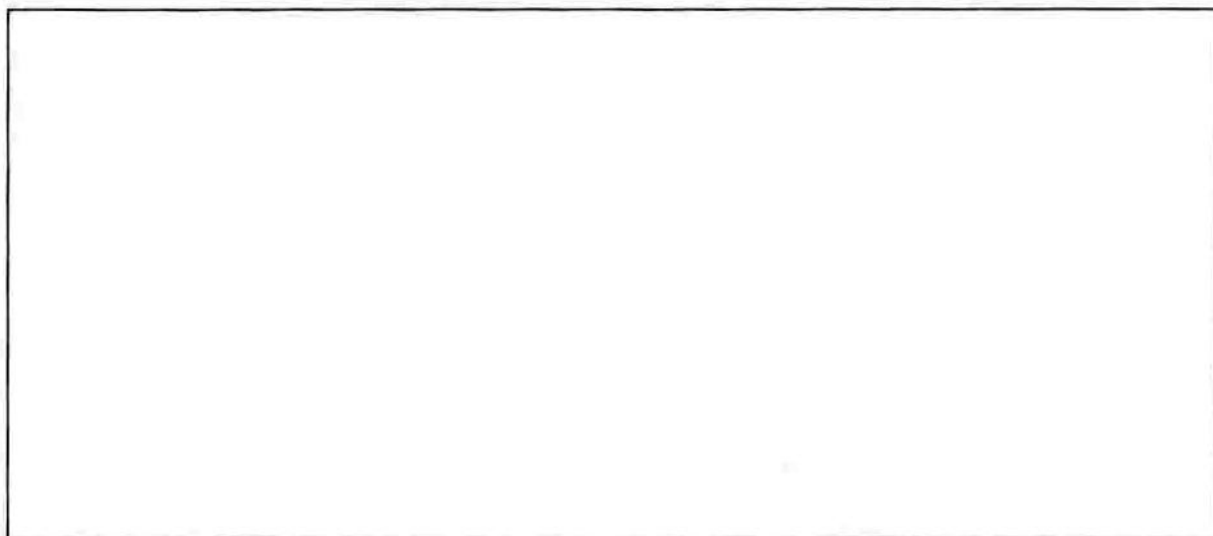
Il **64** funziona con un'accuratezza di circa 8 cifre decimali, il che significa che molte frazioni (ad esempio 1/3 o 1/7) non possono essere rappresentate esattamente. È possibile aspettarsi piccoli errori di arrotondamento in alcuni comandi aritmetici cosicchè il risultato che si pensava fosse esattamente 7, potrebbe risultare come "6.99999998".
Per provare la conoscenza delle espressioni, eseguire i seguenti esempi e prevedere cosa visualizzerà il **64** in ciascun caso. Si supponga che X = 3 e P = 7.

COMANDO	*RISULTATO PREVISTO*	*RISULTATO EFFETTIVO*
PRINT 3 + 12 — 6 — 4		
PRINT 4 + 3 ★ 2		
PRINT X + P — 3		
PRINT 5 + 12 / 6 — 3		
PRINT 11 / 5 — 7 / 4		
PRINT 4 ↑ 2 — 2 ↑ 4		
PRINT 3 + 2 ↑ 3 — 3 ↑ 2		
PRINT 2 ↑ X — P		
PRINT 3 + 12 — (6 — 4)		
PRINT 5 + 12 / (6 — 3)		
PRINT (P + X) ↑ (1 — X)		
PRINT 4 ↑ 2 — 3 ↑ 0		
PRINT (P ↑ 2 — X ↑ 2) / 3		

Ora controllare i risultati sul **64.** Ricordarsi di impostare i valori di P e X prima di iniziare.
Nel BASIC le espressioni sono comunemente usate nei comandi PRINT e LET. Ecco un semplice programma che immette due numeri U e V e visualizza un valore F calcolato secondo la formula:

$$\frac{1}{F} = \frac{1}{V} + \frac{1}{U}, \text{ or } F = \frac{1}{\frac{1}{V} + \frac{1}{U}}$$

```
10 INPUT "V"; V
20 INPUT "U"; U
30 PRINT "F="; 1/(1/V +1/U)
40 STOP
```

Esempio 1
Scrivere un programma che legge due valori V e R e visualizza il valore della formula $A = \frac{V^2}{R}$

Esempio 2
Scrivere un programma che visualizza i valori della formula $y = \frac{1}{1 + x^2}$ per i valori di x compresi fra 0 e 2 procedendo a incrementi di 0.2. (Suggerimento: usare un'iterazione FOR come questa:

```
FOR X = 0 TO 2 STEP 0.2
.....
NEXT X      )
```

(Le risposte effettive sono indicate al termine dell'Appendice B).

2 Funzioni standard

Come la maggior parte dei calcolatori, il **64** dispone di una serie di funzioni "scientifiche". Una particolarmente utile è la radice quadrata, abbreviata in SQR, che può essere inclusa in espressioni tipo questa:

```
PRINT SQR(5)
```

oppure: PRINT SQR(B↑2+C↑2)

La quantità tra parentesi è detta *argomento* della funzione. Nel caso di SQR l'argomento deve essere zero o positivo.
Ecco un programma che visualizza le radici quadrate di tutti i numeri compresi fra 100 e 115.

```
10 PRINT"N"; "SQR(N)"

20 FOR N=100 TO 115

30 PRINT N; SQR(N)

40 NEXT N

50 STOP
```

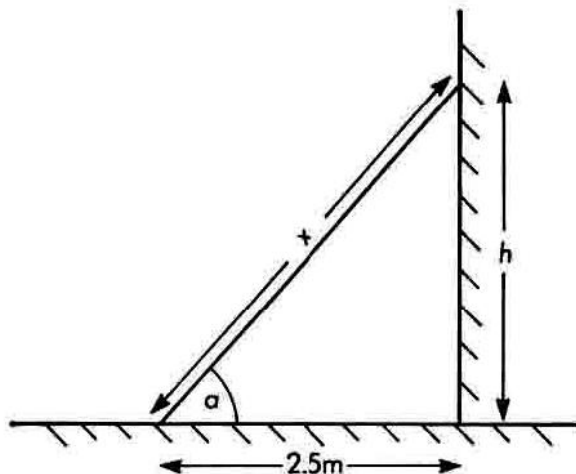
Esempio 3

Se a, b, c sono le lunghezze dei tre lati di un triangolo, l'area del triangolo stesso è indicata dalla formula $a = \sqrt{s\ (s-a)\ (s-b)\ (s-c)}$ dove s è il semiperimetro, (a+b+c)/2.
Scrivere un programma che immette i tre numeri. Se si tratta dei lati di un triangolo reale, il programma visualizza l'area del triangolo; altrimenti, (ad esempio se i numeri sono 1, 1, 10), il programma visualizza un messaggio appropriato.
(Suggerimento: se le righe non formano un triangolo, il valore di s (s−a) (s−b) (s−c) è negativo).

Qui di seguito sono indicate alcune delle più importanti funzioni matematiche. Leggerle ma senza sentirsi obbligati ad impararle a memoria — è sempre possibile fare riferimento all'elenco successivamente.

SIN(X) COS(X) TAN(X)	Funzioni trigonometriche. L'argomento deve essere in radianti (1 grado = $\pi/180$ radianti)
ATN(X)	Arcotangente di X. Il risultato è in radianti compreso tra $-\pi/2$ e $\pi/2$.
LOG(X)	Logaritmo naturale di X (logaritmo in base e) X deve essere positivo
EXP(X)	Equivalente a e^x
ABS(X)	Modulo di X (X se X > 0; altrimenti −X)
INT(X)	Numero intero maggiore, uguale o minore di X. Notare che: INT (3.5) = 3 INT (−3.5) = −4

Controllare la risposta in Appendice B.

È inoltre possibile usare il simbolo di tastiera π invece del numero 3.14149265.....
Ecco un esempio per mostrare l'uso di alcune di queste funzioni. Una scala può variare la sua lunghezza da 4 a 5 metri a intervalli di 20 cm. Essa è disposta con la sua base a 2,5 metri da una parete verticale e con la parte superiore contro la parete. Scrivere un programma per visualizzare l'angolo della scala con l'orizzontale, per ciascuna delle sei possibili lunghezze.
Innanzitutto si svolge il problema matematicamente usando un diagramma. Si userà x per indicare la lunghezza della scala, h come altezza della parte superiore della scala e a come angolo rispetto all'orizzontale.

$h = \sqrt{x^2 - (2.5)^2}$ (secondo Pitagora)

$a = \arctan(h/2.5)$ (in radianti)

o $a = (180/\pi) \star \arctan(h/2.5)$ in gradi

Si scriverà ora il programma, che ha una semplice struttura con iterazioni:

```
10 PRINT "LUNGHEZZA", "ANGOLO"
20 FOR X=4 TO 5 STEP 0.2
30 H = SQR (X↑2−2.5↑2)
40 A = (180/π) ★ ATN (H/2.5)
50 PRINT X, A
60 NEXT X
70 STOP
```

Una delle funzioni più utili è INT. È possibile usarla per dire se un numero divide un altro esattamente. Se X è un multiplo esatto di Y, la condizione

X/Y = INT (X/Y)

sarà vera; altrimenti no.

Un numero è *primo* se non ha divisori salvo se stesso e 1. Il seguente programma calcola e visualizza i numeri primi da 3 a qualsiasi valore definito dall'utente:

```
10 INPUT "VALORE PIÙ ELEVATO"; H
20 FOR N = 3 TO H
30 FOR J=2 TO N−1
40 IF N/J=INT(N/J) THEN 70
50 NEXT J
60 PRINT N;
70 NEXT N
80 STOP
```

Esempio 4

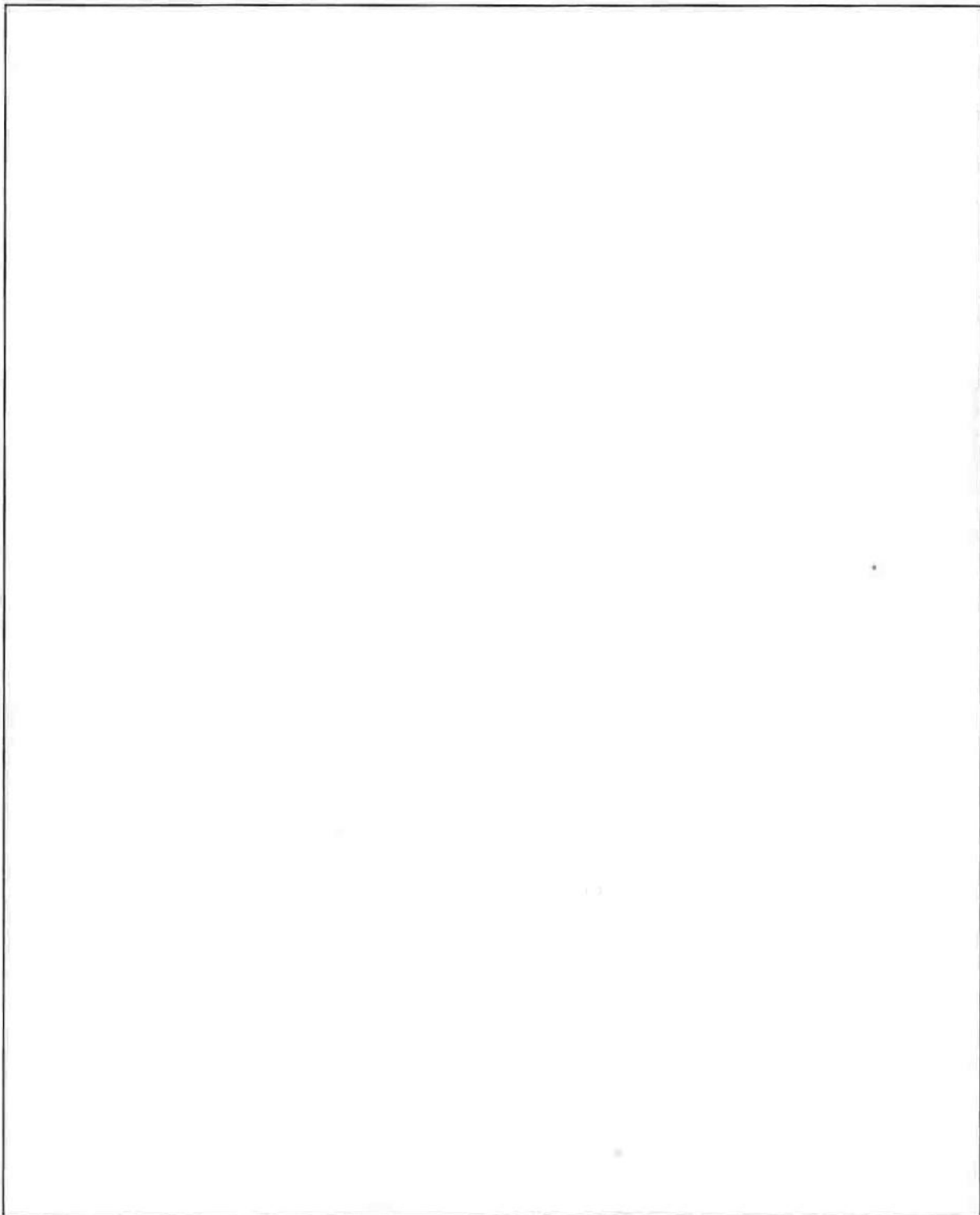
Studiare il programma dei numeri primi (controllandolo se necessario) e scoprire come funziona. Eseguirlo e provarlo per qualche valore di H (ad esempio 500).

Questo metodo di calcolo dei numeri primi è effettivamente molto lento. Disegnare e incorporare alcuni miglioramenti per farlo girare più velocemente.

Suggerimenti:

(a) Nessun numero pari salvo 2 può essere primo.

(b) Nella prova di possibili fattori è sufficiente arrivare fino alla radice quadrata del numero.

Controllare ora la risposta nell'Appendice B.

APPENDICE B

UNITA':7

Esperimento 7.1:

a) T,T,T,T,F,F,F
b) F,F,T,T

Esperimento 7.3:

(1)
```
10 P$= "*"
20 PRINT P$
30 P$= P$+"*"
40 IF P$<> "***********"
   THEN 20
50 STOP
```

(2)
```
10 PRINT "POUNDS", "DOLLARS"
20 PRINT
30 P= 10
40 PRINT P, 1.77*P
50 P=P+2
60 IF P< 32 THEN 40
70 STOP
```

(3)
```
10 PRINT "CENT", "FAHR"
20 PRINT
30 C= 15
40 F= 1.8*C+32
50 PRINT C, F
60 C=C+1
70 IF C < 31 THEN 40
80 STOP
```

Esperimento 7.2:

Variabile di controllo	*Valore iniziale*	*Valore finale*	*Incremento*	*Numero di iterazioni*
X$	"A"	"ABBB"	"B"	4
P	0	10	+1	11
Y$	"Z"	"ZXYXY"	"XY"	3
R	5	14	3	4
C	27	7	–5	5

UNITA':8

Esperimento 8.1:

a) CONTATORE DI PROGRAMMA ~~10~~ ~~20~~ ~~30~~ ~~40~~ ~~50~~ 60

VARIABILI X: 5 Y: 7 Z: 12 W: 2

5 7 12 2 BREAK IN 60 READY	10 X=5 20 Y=7 30 Z=X+Y 40 W=Y−X 50 PRINT X; Y; Z; W 60 STOP

CONTATORE DI PROGRAMMA ~~10~~ ~~20~~ ~~30~~ ~~40~~ ~~50~~ ~~30~~ ~~40~~ ~~50~~ 60

VARIABILI Q: ~~1~~ ~~2~~ 3

LEI MI AMA LEI NON MI AMA LEI NON MI AMA BREAK IN 60 READY	10 Q=1 20 PRINT"LEI MI AMA" 30 PRINT"LEI NON MI AMA" 40 Q=Q+1 50 IF Q < 3 THEN 30 60 STOP

Esperimento 8.2:

a) La riga 50 dovrebbe intendersi:
50 IF G < 11 THEN 30

b) La riga 30 dovrebbe intendersi:
30 A$=A$+"★"

Esperimento 8.3:

c) Riga 20: PRINT (non PRINT)
Nessun RETURN dopo la riga 40
Riga 60: IF X < 13 THEN 40 (NOT X > 13)
Riga 70: STOP (Non STØP)

UNITA':9

Esperimento 9.2:

```
10 POKE 53280,4
15 POKE 53281,7
20 PRINT " [SHIFT] e [CLR HOME] ";
30 PRINT " [CLR HOME] [CRSR] ←9 volte→ [CRSR]
   [CRSR] ←6 volte→ [CRSR] ";
40 PRINT TI$
50 GOTO 30
```

Esperimento 9.3:

```
5 REM BANDIERA DELL'ISLANDA
10 POKE 53280,14
15 POKE 53281,6
20 PRINT " [SHIFT] e [CLR HOME] ";
30 J=1
40 PRINT " [CTRL] e [RVS ON]
   [CRSR] ←12 volte→ [CRSR] [CTRL] e
   [WHT] ←1 spazio→ [CTRL] e [RED]
   ← 4 spazi → [CTRL] e [WHT] 1 spazio"
50 J=J+1
60 IF J<11 THEN 40
70 PRINT " [CTRL] e [RVS ON] [CTRL] e
   [WHT] ← 13 spazi → [CTRL] e [RED]
   ← 4 spazi → [CTRL] e [WHT]
   ← 23 spazi →";
80 J=1
90 PRINT " [CTRL] e [RVS ON] [CTRL] e
   [RED] ← 40 spazi →";
100J=J+1
110 IF J<4 THEN 90
120 PRINT " [CTRL] e [RVS ON] [CTRL] e
   [WHT] ← 13 spazi → [CTRL] e [RED]
   ← 4 spazi → [CTRL] e [WHT]
   ← 23 spazi →";
130 J=1
140 PRINT " [CTRL] e [RVS ON]
   [CRSR] ←12 volte→ [CRSR]
   [CTRL] e [WHT] ←1 spazio→
   [CTRL] e [RED] ← 4 spazi →
   [CTRL] e [WHT] ←1 spazio→"
150 J=J+1
160 IF J<10 THEN 140
170 PRINT " [CTRL] e [RVS ON]
   [CRSR] ←12 volte→ [CRSR]
   [CTRL] e [WHT] ←1 spazio→
   [CTRL] e [RED] ← 4 spazi →
   [CTRL] e [WHT] ←1 spazio→";
180 GOTO 180
```

UNITA':10

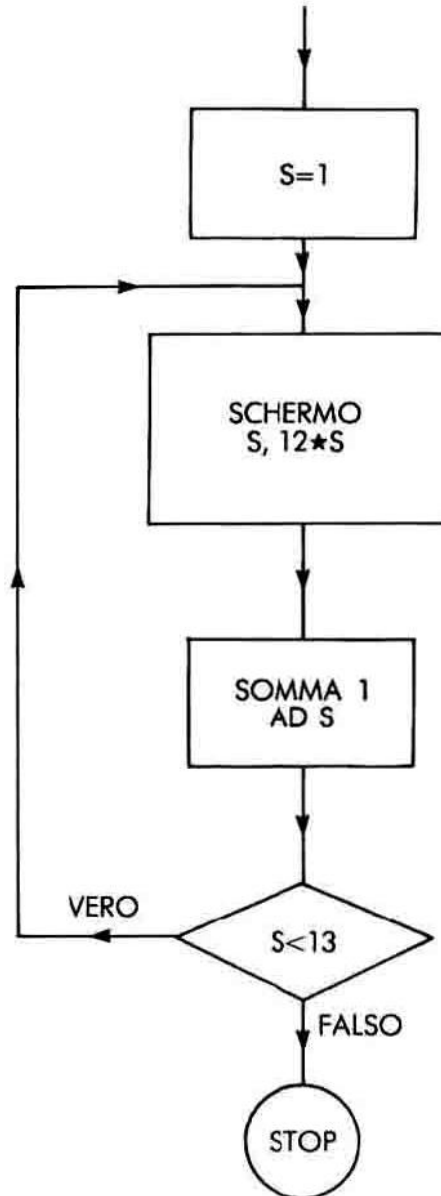
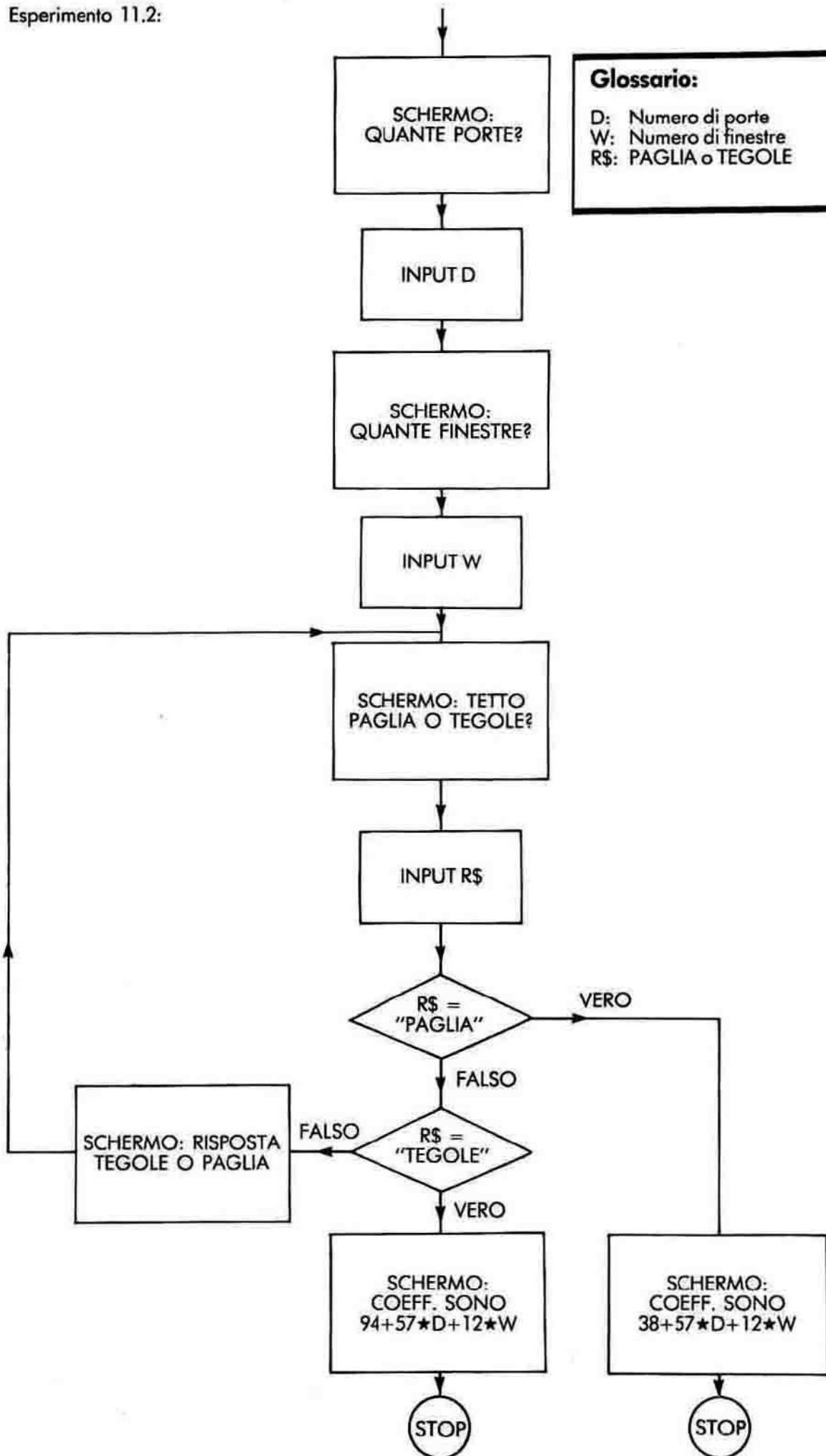
Esperimento 10.2:

a)
```
10 INPUT "PROGRAMMA TABELLINE"
20 INPUT "PER"; N
30 X=1
40 PRINT X; "PER"; N; "FA"; N*X
50 X=X+1
60 IF X<13 THEN 40
70 STOP
```

b)
```
10 PRINT "DIMMI IL TUO"
20 INPUT "COGNOME" S$
30 PRINT "DIMMI IL NOME"
40 INPUT "DI TUA MOGLIE"; C$
50 PRINT "IL SUO NOME COMPLETO È"
60 PRINT C$+ "    "+S$
70 STOP
```

UNITA':11

Esperimento 11.1:

Esperimento 11.2:

```
10 REM COEFFICIENTI RURITANIA
20 PRINT "PROGRAMMI VALUTAZIONE"
30 INPUT "QUANTE PORTE"; D
40 INPUT "QUANTE FINESTRE"; W
50 PRINT "TETTO"
60 PRINT "TEGOLE O"
70 INPUT "PAGLIA"; R$
80 IF R$="PAGLIA" THEN 140
90 IF R$="TEGOLE" THEN 160
100 PRINT "RISPONDI"
110 PRINT "TEGOLE O"
120 PRINT "PAGLIA"
130 GOTO 50
140 PRINT "COEFF SONO"; 38+57*D+12W
150 STOP
160 PRINT "COEFF SONO"; 94+57*D+12W
170 STOP
```

Le risposte corrette ai tre problemi campione, sono:

a) 95 b) 155 c) 364

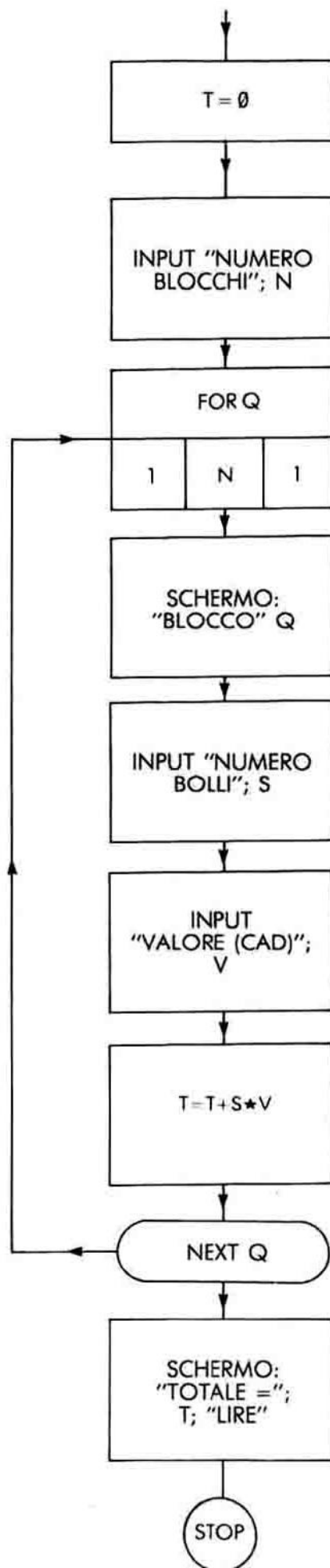
UNITA':12

Esperimento 12.1:

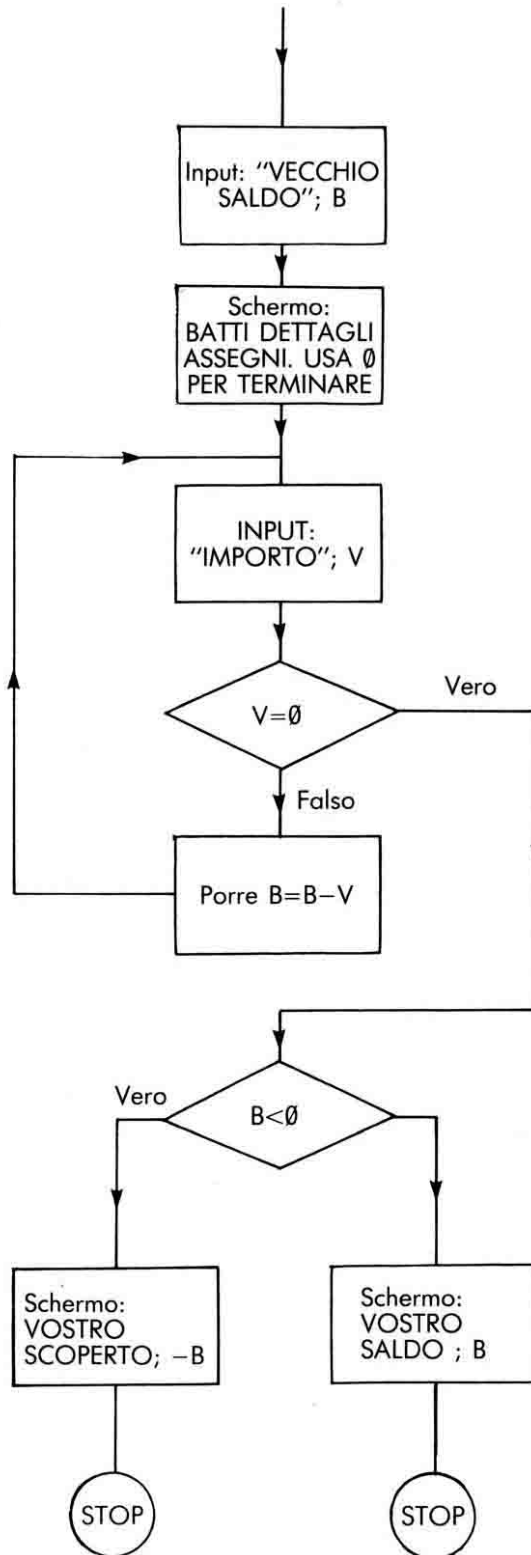
```
10 RS=0
20 INPUT "NUMERO INNINGS"; J
30 FOR Q=1 TO J
40 INPUT "PUNTEGGIO"; S
50 RS=RS+S
60 NEXT Q
70 PRINT "MEDIA="; RS/J
80 STOP
```

Esperimento 12.2:

Glossario

N: Numero dei blocchi
S: Numeri dei bolli in un blocco
V: Valore di ciascun bollo in un blocco
T: Totale mobile
Q: Per contare il numero dei blocchi

```
 10 T=0
 20 INPUT "NUMERO BLOCCHI; N
 30 FOR Q=1 TO N
 40 PRINT "BLOCCO"; Q
 50 INPUT "NUMERO BOLLI"; S
 60 INPUT "VALORE (CADAUNO)"; V
 70 T=T+S★V
 80 NEXT Q
 90 PRINT "TOTALE="; T; "LIRE"
100 STOP
```

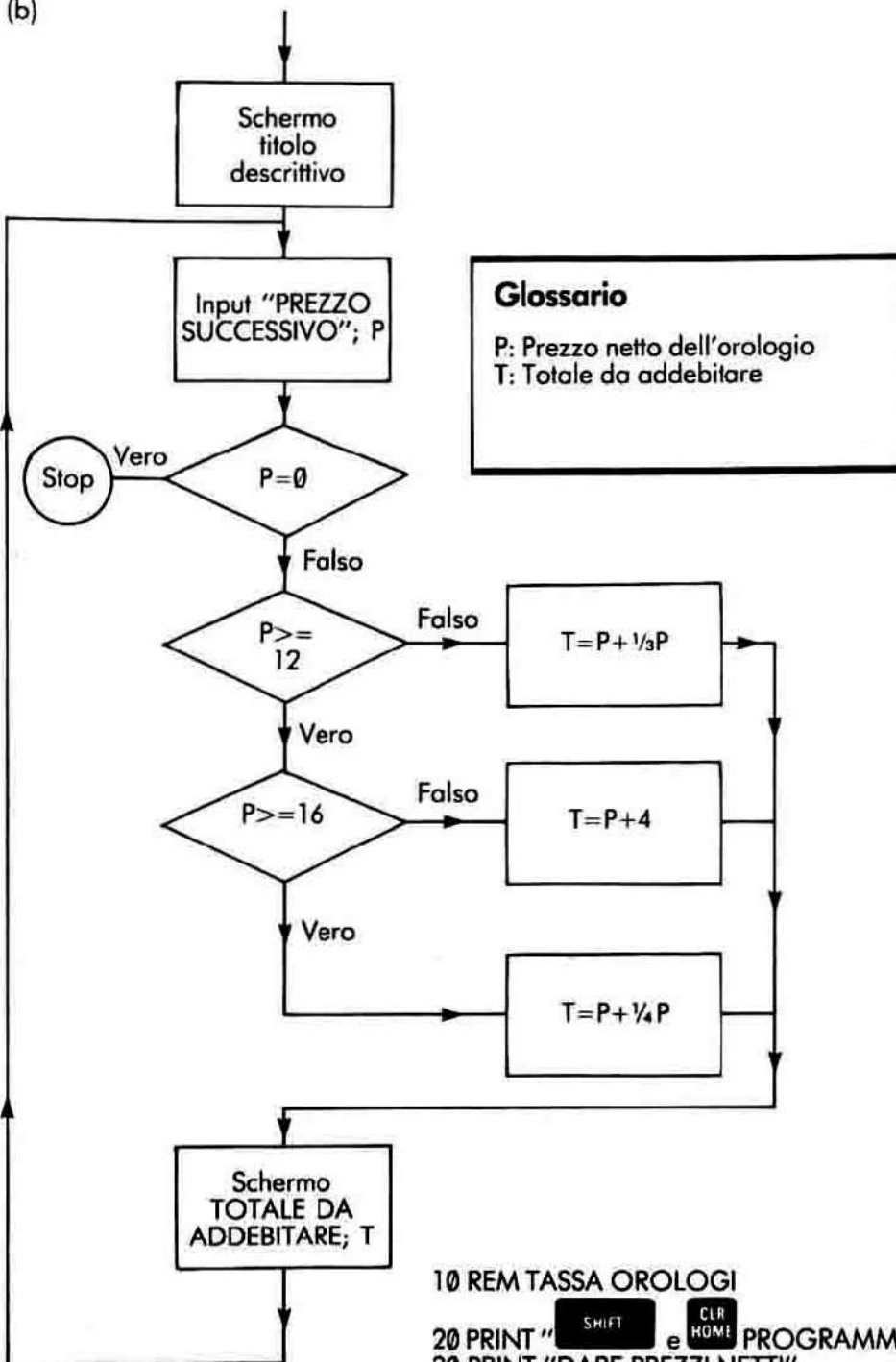
UNITA':14

Esperimento 14.1:

Glossario:

B: Saldo corrente
V: Ciascuna nuova transazione

```
 10 REM PROGRAMMA BANCARIO
 20 INPUT "VECCHIO SALDO"; B
 30 PRINT "BATTI DETTAGLI ASSEGNI"
 40 PRINT "USA 0 PER TERMINARE"
 50 INPUT "IMPORTO"; V
 60 IF V=0 THEN 90
 70 B=B-V
 80 GOTO 50
 90 IF B<0 THEN 120
100 PRINT "VOSTRO SALDO LIRE"; B
110 STOP
120 PRINT "VOSTRO SCOPERTO"
130 PRINT "LIRE"; -B
140 STOP
```

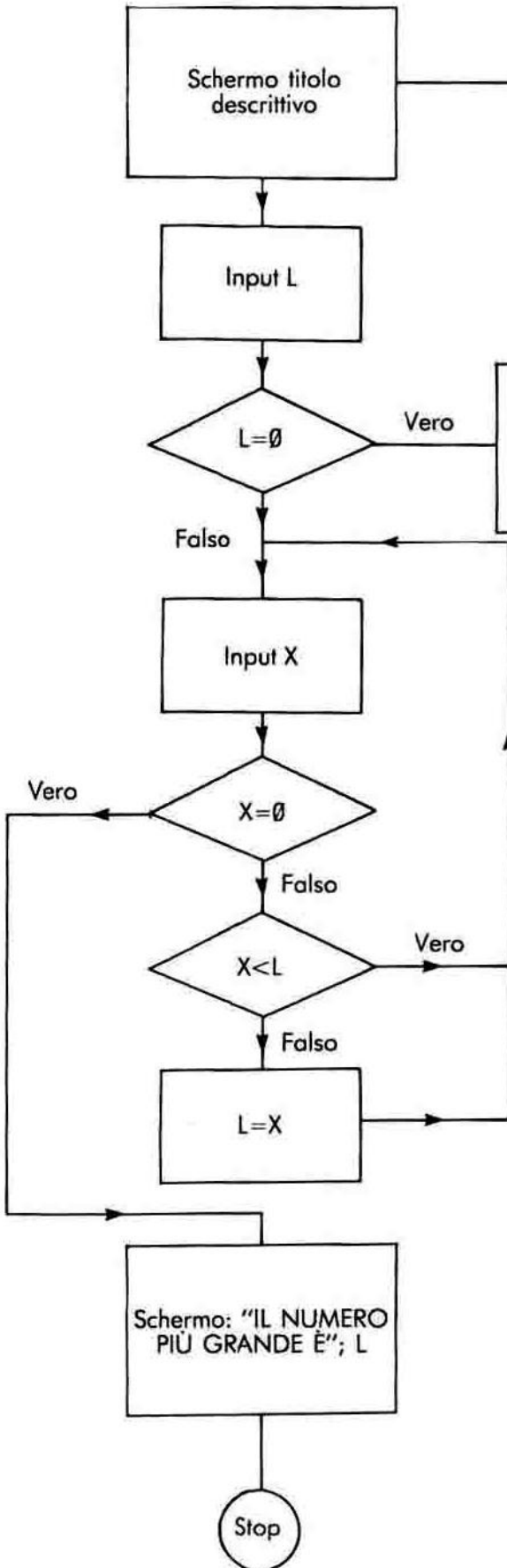
Esperimento 14.2:

(b)

Glossario

P: Prezzo netto dell'orologio
T: Totale da addebitare

```
10 REM TASSA OROLOGI
20 PRINT "[SHIFT] e [CLR HOME] PROGRAMMA TASSA OROLOGI"
30 PRINT "DARE PREZZI NETTI"
40 PRINT "USA Ø PER TERMINARE"
50 INPUT "PREZZO SUCCESSIVO"; P
60 IF P=Ø THEN 18Ø
70 IF P>=12 THEN 1ØØ
80 T=P+(1/3)★P
90 GOTO 14Ø
100 IF P>=16 THEN 13Ø
110 T=P+4
120 GOTO 14Ø
130 T=P+(1/4)★P
140 PRINT "TOTALE DA ADDEBITARE"
150 PRINT "È"; T
160 PRINT
170 GOTO 5Ø
180 STOP
```

Esperimento 14.2:

c)

Glossario:

L: Numero più grande fino a questo momento
X: Successivo numero da immettere

```
10 REM INDICA NUMERO PIÙ GRANDE
20 PRINT "DAMMI I NUMERI TERMINATI"
30 PRINT "DA 0"
40 INPUT "IL PROSSIMO"; L
50 IF L=0 THEN 130
60 INPUT "IL PROSSIMO"; X
70 IF X=0 THEN 110
—0 IF X<L THEN 60
90 L=X
100 GOTO 60
110 PRINT "IL PIÙ GRANDE È"; L
120 STOP
130 PRINT "DAMMI ALMENO UN NUMERO"
140 PRINT "DIVERSO DA ZERO"
150 GOTO 20
```

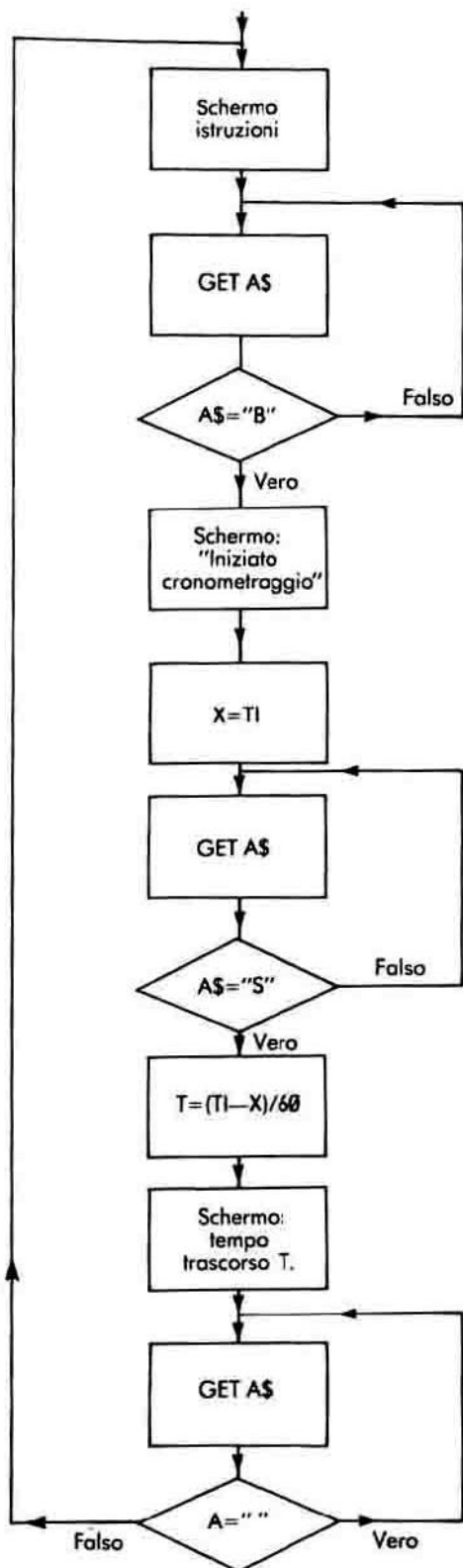
(Nota: la seguente "soluzione" non funziona se tutti i numeri sono negativi — e cioè inferiori a 0).

```
10 L=0
20 INPUT "IL PROSSIMO"; X
30 IF X=0 THEN 70
40 IF X<L THEN 20
50 L=X
60 GOTO 20
70 PRINT "IL PIÙ GRANDE È"; L
80 STOP
```

Perchè no?

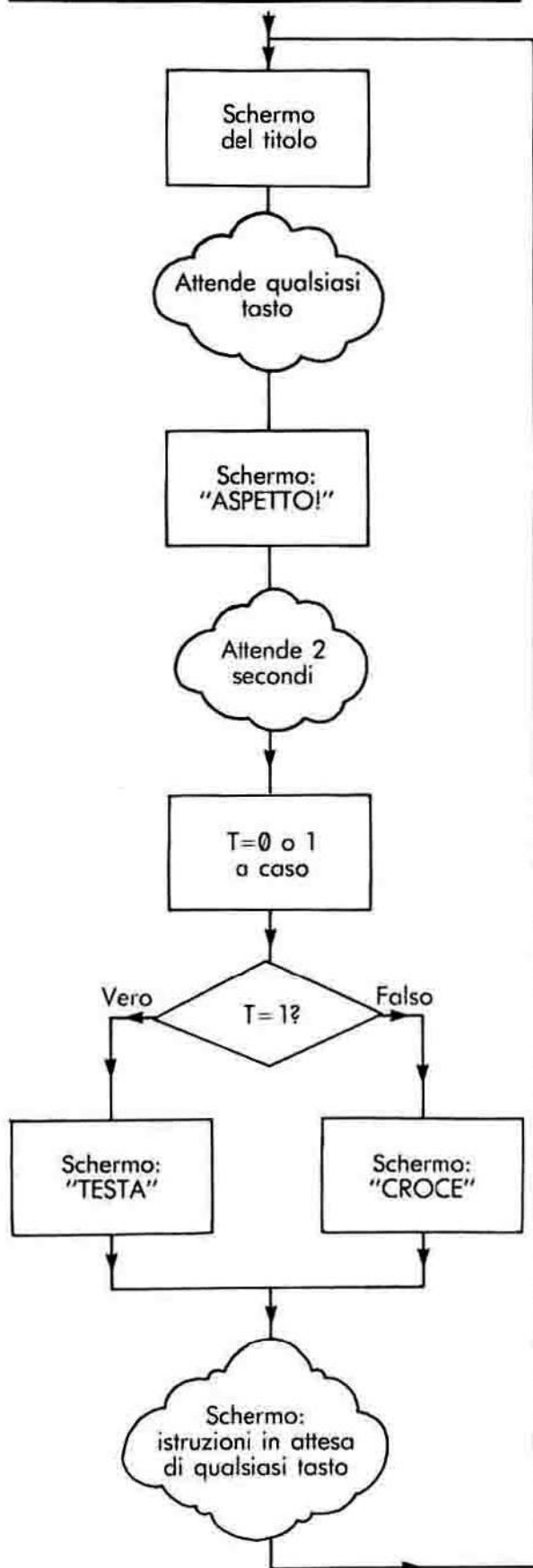
UNITA':15

Esperimento 15.2(1):

Schermo istruzioni
GET A$
A$="B"
Falso
Vero
Schermo: "Iniziato cronometraggio"
X=TI
GET A$
A$="S"
Falso
Vero
T=(TI—X)/60
Schermo: tempo trascorso T.
GET A$
A=" "
Falso
Vero

Glossario:

A$: Caratteri da tastiera
X: Tempo interno all'inizio dell'intervallo (jiffies)
T: Tempo trascorso (secondi)

```
  5 REM CRONOMETRAGGIO
 10 PRINT "[SHIFT] e [CLR HOME]"
 20 PRINT "PROGRAMMA
    CRONOMETRAGGIO"
 30 PRINT
 40 PRINT "PER INIZIARE CRONOMETRAGGIO"
 50 PRINT "BATTI IL TASTO B"
 60 PRINT "PER INTERROMPERE BATTI S"
 70 GET A$
 80 IF A$ <> "B" THEN 70
 90 X=TI
 95 PRINT "TEMPO DI PARTENZA"
100 GET A$
110 IF A$ <> "S" THEN 100
120 T=(TI—X)/60
130 PRINT "TEMPO TRASCORSO"
140 PRINT T; "SECONDI"
150 PRINT
160 PRINT "ORA BATTI UN QUALSIASI
    ALTRO TASTO"
170 PRINT "PER UN ALTRO
    CRONOMETRAGGIO"
180 GET A$
190 IF A$=" " THEN 180
200 GOTO 10
```

Esperimento 15.2(2):

Glossario:

S$: Carattere da tastiera
M: Usato nell'iterazione per attendere 2 secondi
T: 0 (per croce) o 1 (per testa)

```
 10 REM LANCIO MONETA
 20 PRINT "SHIFT e CLR HOME"
 30 PRINT "BATTI QUALSIASI TASTO PER"
 40 PRINT "LANCIARE MONETA"
 50 GET S$
 60 IF S$="" THEN 50
 70 PRINT "ASPETTA!"
 80 PRINT
 90 FOR M=1 TO 2000
100 NEXT M
110 T=INT (0+2*RND(0))
120 IF T=1 THEN 150
130 PRINT CROCE
140 GOTO 160
150 PRINT "TESTA"
160 PRINT
170 PRINT "BATTI QUALSIASI TASTO PER"
180 PRINT "PROCEDERE"
190 GET S$
200 IF S$="" THEN 190
210 GOTO 20
```

Esperimento 15.4:

```
READY

   5 REM CRAPS
  10 VV = 212 * 256
  15 POKE VV + 24,15
  20 POKE 53281,1
  25 POKE VV + 6,0
  30 PRINT" [SHIFT] and [CLR HOME] [C=] and 5"
  40 PRINT" THE GAME OF CRAPS IS PLAYED
     WITH"
  50 PRINT" TWO DICE. FIRST YOU BET AND
     THEN YOU
  60 PRINT" THROW. IF YOU GET A SCORE
     OF 7 OR 11,
  70 PRINT "YOU WIN. IF YOU THROW 2,3
     OR 12, YOU
  80 PRINT "LOSE. IF YOU THROW ANY
     OTHER NUMBER,
  90 PRINT "YOU DON'T WIN OR LOOSE
     STRAIGHT AWAY;
 100 PRINT "YOU KEEP ON THROWING
     UNTIL YOU
 110 PRINT "THROW THE SAME AS YOU DID
     FIRST
 120 PRINT "TIME (AND WIN)
 130 PRINT "OR
 140 PRINT "THROW A 7 (AND LOSE).
 150 PRINT
 160 PRINT "HIT ANY KEY TO CONTINUE
 240 GET A$
 250 IF A$ = " " THEN 240
 255 REM SET A$, B$, C$, TO LINES OF DICE
     PICTURE
 260 A$ = "← 4 spaces → ╭───╮
     ← 2 spaces → ╭───╮ "
 270 B$ = "← 4 spaces → | ← 3 spaces →
     | ← 2 spaces → | ← 3 spaces → | "
 280 C$ = "← 4 spaces → ╰───╯
     ← 2 spaces → ╰───╯ "
 285 REM GET STARTING CAPITAL
 290 PRINT" [SHIFT] and [CLR HOME] "
 300 INPUT "STARTING CAPITAL"; C
 305 REM NOW START NEXT BET
 310 PRINT "HIT ANY KEY FOR NEXT BET"
 330 GET R$
 340 IF R$= " "THEN 330
 350 PRINT "YOUR CAPITAL NOW IS"; C
 370 INPUT "HOW MUCH DO YOU BET"; W
 390 IF W <= C THEN 420
 400 PRINT "YOU CAN'T AFFORD IT"
 410 GOTO 310
 415 REM ORGANISE FIRST THROW
 420 PRINT" [SHIFT] and [CLR HOME] [CRSR] [CRSR] [CRSR]
     ←3 spaces→ FIRST THROW (BET = "; W;")"
 430 PRINT " [CLR HOME] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR]
     [CRSR] "; A$
 440 FOR J= 1 TO 5
 450 PRINT B$
 460 NEXT J
 470 PRINT C$
 475 REM SHOW 10-59 DIFFERENT FACE PAIRS
 480 Q= INT (10 +50 * RND (0))
 490 FOR Z = 1 TO Q
 500 A= INT (1+6*RND (0))
 510 B= INT (1+6*RND(0))
 515 REM SOUND A NOTE WHICH DEPENDS
     ON A AND B
 520 POKE VV+4,0
 525 POKE VV+5,7+1,10
 530 POKE VV+,10 +(A*A+2*B*B)
 535 POKE VV+4,33
 540 PRINT " [CLR HOME] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR]
     [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR]
     [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] "; A; "
     [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR]
     [CRSR] [CRSR] "; B
 545 REM WAIT A BIT
 550 FOR M=1 TO 30
 555 NEXT M
 560 NEXT Z
 565 REM STOP SOUND
 570 POKE VV+4,0
 585 REM USE LAST VALUES OF A, B
 590 T=A+B
 595 REM JUMP IF PLAYER WINS OUTRIGHT
 600 IF T=7 THEN 1000
 610 IF T=11 THEN 1000
 615 REM JUMP IF PLAYER LOSES OUTRIGHT
 620 IF T=2 THEN 1100
 630 IF T=3 THEN 1100
 640 IF T=12 THEN 1100
 650 PRINT
 660 PRINT
 670 PRINT
 680 PRINT "YOU HAVE TO MAKE"; T;
     "BEFORE 7"
 700 PRINT " [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR] [CRSR]
     [CRSR] ←3 spaces→ HIT ANY KEY TO GO ON"
 710 GET R$
 720 IF R$= " "THEN 710
 730 PRINT " [SHIFT] and [CLR HOME] [CRSR] ←3 spaces→
     NEXT THROW (BET="; W;")"
 740 PRINT "← 3 spaces → MAKING"; T
 750 PRINT " [CLR HOME] [CRSR] [CRSR] [CRSR] [CRSR] "
 760 PRINT A$
 770 FOR J=1 TO 5
 780 PRINT B$
 790 NEXT J
 800 PRINT C$
 805 REM SHOW 10-19 DIFFERENT FACE
     PAIRS
 810 Q= INT (10 +10 *RND(0))
```

```
820 FOR Z=1 TO Q
830 A=INT (1+6*RND(0))
840 B=INT (1+6*RND(0))
845 POKE VV+4,0
850 POKE VV+5,7
855 POKE VV+1,10 +3*(A*A+B*B)
860 POKE VV+4,33
870 PRINT "[CLR HOME][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR]
    [CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR]
    [CRSR][CRSR][CRSR][CRSR][CRSR]"; A;
   "[CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR]
    [CRSR][CRSR]"; B
880 FOR M=1 TO 30
890 NEXT M
900 NEXT Z
905 REM SILENCE
910 POKE VV+4,0
925 REM IF A+B=T PLAYER WINS
930 IF A+B=T THEN 1000
935 REM IF A+B=7 PLAYER LOSES
940 IF A+B=7 THEN 1100
945 REM ELSE PLAYER THROWS AGAIN
950 GOTO 700
990 REM PLAYER WINS
1000 PRINT "[CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR]
← 3 spaces → YOU WIN"
1005 REM ADD WINNINGS TO CAPITAL
1010 C=C+W
1015 REM PAEAN OF PRAISE
1020 POKE VV+1,110
1025 POKE VV+5,8
1030 POKE VV+6,0
1040 FOR J=1 TO 8
1050 POKE VV+4,33
1060 FOR K=1 TO 70
1065 NEXT K
1070 POKE VV+4,0
1075 FOR K=1 TO 80
1080 NEXT K
1085 NEXT J
1095 GOTO 310
1100 REM PLAYER LOSES
1110 PRINT "[CRSR][CRSR][CRSR][CRSR][CRSR][CRSR][CRSR]
    [CRSR][CRSR][CRSR] ← 3 spaces → YOU LOSE"
1115 REM CHIRP OF TRIUMPH"
1120 POKE VV+5,0
1125 POKE VV+6,240
1130 POKE VV+1,100
1140 POKE VV+4,33
1150 FOR J=100 TO 5 STEP -0.3
1160 POKE VV+1,J
1170 NEXT J
1180 POKE VV+4,0
1190 POKE VV+6,0
1195 REM TAKE LOSS FROM CAPITAL
1200 C=C-W
1210 IF C>0 THEN 310
1220 PRINT "YOU ARE NOW BROKE"
1230 STOP
```

APPENDICE A
SOLUZIONE AI PROBLEMI

Esempio 1:

```
10 INPUT V
20 INPUT R
30 PRINT "A="; V↑2/R
40 STOP
```

Esempio 2:

```
10 PRINT "X  FORMULA"
20 FOR X=0 TO 2 STEP 0.2
30 PRINT X; 1/(1+X↑2)
40 NEXT X
50 STOP
```

Esempio 3:

```
 10 PRINT "INDICARE I TRE LATI"
 20 INPUT "A"; A
 30 INPUT "B"; B
 40 INPUT "C"; C
 50 S= (A+B+C)/2
 60 X= S*(S—A)*(S—B)*(S—C)
 70 IF X <0 THEN 100
 80 PRINT "L'AREA È"; SQR(X)
 90 STOP
100 PRINT "NON SONO"
110 PRINT "LATI DI UN TRIANGOLO"
120 STOP
```

Glossario:

A,B,C: Tre "lati" del triangolo
S: Semi-perimetro
X: Quadrato dell'area (eventuale)

Esempio 4:

```
 10 REM VERSIONE LEGGERMENTE PIÙ VELOCE
 20 INPUT "VALORE PIÙ ALTO"; H
 30 FOR N=3 TO H STEP 2
 40 Q= SQR (N)
 50 FOR J=2 TO Q
 60 IF N/J= INT (N/J) THEN 90
 70 NEXT J
 80 PRINT N;
 90 NEXT N
100 STOP
```

APPENDICE C

Messaggi di errore

Questo elenco illustra gli errori che possono insorgere se si usano le funzioni BASIC descritte in questo manuale. Altri errori possono verificarsi se si eseguono programmi di natura più avanzata.

Division by Zero
(Divisione per zero)

La divisione di un numero per zero non è ammessa. Questo errore può insorgere nei comandi tipo

```
10 A = 5/0
```

oppure
```
20 B = Q/(J−J)
```

Extra Ignored
(Ignorati gli extra)

Se si battono troppe voci (numeri o stringhe) in risposta ad un comando INPUT, quelle extra vengono ignorate. Il programma non si interrompe.

Illegal quantity
(Quantità illecita)

Un numero usato in un comando è troppo ampio (o troppo piccolo). Per esempio qualsiasi numero che si inserisce (POKE) in una locazione, deve essere nel campo da 0 a 255.
Questo errore può verificarsi in comandi tipo

```
10 POKE 36878, 1234
```

oppure
```
20 J = 300
30 POKE 36876, J
```

Load Error
(Errore di caricamento)

Il programma non viene caricato correttamente dal registratore a cassetta. Cercare di pulire la testina di lettura. Alternativamente il programma può non essere stato registrato correttamente all'origine oppure il nastro può essere stato danneggiato da un campo magnetico.

Next Without For
(Next senza For)

La struttura FOR-NEXT del programma è sbagliata.

Out of Memory
(Manca memoria)

Il computer ha esaurito lo spazio di memoria. Ciò si verifica soltanto con programmi molto lunghi o che usano grandi quantità di dati.

Redo from Start
(Ripartire da capo)

Se un comando INPUT aspetta un numero e si batte qualcosa di diverso, il computer visualizza questo messaggio e consente di ripartire da capo.

String Too Long
(Stringa troppo lunga)

Una stringa formata mediante concatenamento è superiore a 255 byte.

Syntax Error
(Errore di sintassi)

Un "comando" ha violato le regole della grammatica BASIC. Le possibili cause sono parentesi non accoppiate, parole chiave non correttamente scritte o elementi di espressioni nell'ordine sbagliato.

Type Mismatch
(Errato accoppiamento di tipo)

Ciò significa che è stato usato un numero invece di una stringa o viceversa.

Verify Error
(Errore in verifica)

Il processo di verifica non è riuscito. Provare a salvare (SAVE) il programma di nuovo.

INDICE ALFABETICO

INDICE ALFABETICO

INTRODUZIONE AL BASIC PARTE 1

NASTRO 1

TESTCARD
HANGMAN
SPEEDTYPE
UNIT3QUIZ
UNIT4DRILL
UNIT5QUIZ

NASTRO 2

SENTENCES
UNIT7QUIZ
UNIT8PROG
UNIT9QUIZ
UNIT10QUIZ
UNIT11PROG
UNIT12QUIZ
SOUND DEMO
HEADS
REACTION
ESG
CRAPS

NOTE: Tutti i programmi sono registrati in tutti i due lati del nastro.